P. 1877/86

Nowy adres redakcji
04-517 Warszawa, ul. Mickiewicza 18 m. 17
tel.: 39-11-34

6 1986

informatyka

Język MML

Transakcje w rozproszonych bazach danych

Logo ze stacją dyskietek

Grafika żółwia w języku Forth

WYDAWNICTWO NOT SIGMA

Cena 120 zł

informatyka

6 1986

Nr 6
Miesięcznik Rok XXI
Czerwiec 1986

Organ Komitetu Informatyki MNSZWiT oraz Komitetu Naukowo-Technicznego NOT ds. Informatyki

KOLEGIUM REDAKCYJNE:

Dr inż. Wacław ISZKOWSKI, mgr Teresa JABŁOŃSKA (sekretarz redakcji), Władysław KLEPACZ (redaktor naczelny), dr inż. Marek MACHURA, Maria PAWLAK (sekretarz redakcji), mgr inż. Jan RYŻKO, dr inż. Wiktor RZECZKOWSKI, dr inż. Janusz ZALEWSKI (zastępca red. naczelnego)

STALE WSPÓŁPRACUJĄ:

Mgr inż. Witold ABRAMOWICZ (Szwajcaria), mgr inż. Teresa WILCZEK

PRZEWODNICZĄCY RADY PROGRAMOWEJ:

Prof. dr hab. Juliusz Lech KULIKOWSKI

Materiałów nie zamówionych redakcja nie zwraca

Redakcja: 01-517 Warszawa, ul. Mickiewicza 18 m. 17, tel. 39-11-34

Zakł. Graf. „Tamka". Zam. 0468-1300/86. Obj. 4,0 ark. druk. Nakład 8300 egz. P-62.

ISSN 0542-9951, INDEKS 36124

Cena egzemplarza 120 zł
Prenumerata roczna 1200 zł

WYDAWNICTWO NACZELNA ORGANIZACJA TECHNICZNA
SIGMA
CZASOPISM I KSIĄŻEK TECHNICZNYCH
00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE: Strona

MML — język komunikacji z systemem komutacyjnym
Władysław Udrycki 1

Generowanie planów wykonywania transakcji w systemach rozproszonych baz danych (1). Podstawowe pojęcia i założenia
Zbyszko Królikowski 4

Współpraca Logo Sinclaira ze stacją dyskietek
Zdzisław Płoski 9

Grafika żółwia w języku Forth
Zbigniew Szkaradnik 11

Mikrokomputerowy system operacyjny SI (2)
Krzysztof Perycz 14

Rola i zakres formalnej definicji Ady (2)
Dines Bjørner 17

Technologia wytwarzania oprogramowania z komputerowym wspomaganiem
Piotr Zaskórski, Ewa Kasprzak 20

Z KRAJU 23

Konkurs Polskiego Towarzystwa Informatycznego na najlepsze prace magisterskie z informatyki — rozstrzygnięty

Szybki system operacyjny DOS

ZE ŚWIATA 24

Strukturalizacja Basica
Kto jest kim w IFIP? George Glaser
Transputery — przyszłość techniki komputerowej?
Sytuacja na rynku elementów półprzewodnikowych
Megabitowe kostki pamięciowe

RECENZJE 29

Ethernet — księga źródłowa

TERMINOLOGIA 30

Terminologia definicji formalnej Ady

W NAJBLIŻSZYCH NUMERACH:

- Vasilii Zakharov o komputerach osobistych i indywidualnych stanowiskach roboczych
- Roman Grabowicz i Janusz Zalewski o systemie operacyjnym
- Jan Zabrodzki o stanie obecnym i perspektywach rozwoju układów LSI i VLSI
- Edmund Pierzchała o interakcyjnym środowisku języka LISP
- Jacek Owczarczyk i Maciej Stolarski o architekturze współbieżnych systemów przetwarzania obrazów
- Wacław Iszkowski o architekturach systemów n-mikroprocesorowych
- Lesław Budzianowski, Jerzy Wietrzych i Jan Kwiatkowski o komputerze Odra 1305 w Międzyuczelnianej Sieci Komputerowej

WŁADYSŁAW UDRYCKI
Instytut Telekomunikacji
Politechnika Warszawska

P. 1877/86

MML — język komunikacji z systemem komutacyjnym

Międzynarodowy Komitet Doradczy Telefonii i Telegrafii (CCITT) wydał zbiór zaleceń dotyczących sposobu komunikowania się operatora z systemem komutacyjnym. W nowo utworzonym języku MML (ang. Man-Machine Langauge — język komunikacji człowiek-maszyna) zawarto zasady leksykalne i procedury wzajemnej wymiany informacji. Język powstał w związku ze wzrostem złożoności funkcjonalnej systemów komutacyjnych i ich sieci, koniecznością stworzenia dostępu do „wnętrza" systemu, jak i potrzebą standaryzacji zasad współpracy operatora z systemem. MML wykracza daleko poza samą specyfikację komend współpracy. Tworzy bogaty opis styku z systemem dla wszystkich funkcji z nim związanych. Język jest tak pomyślany, aby był łatwo przyswajalny, zwiększał komunikatywność systemu i zapewniał pomoc operatorowi w każdym momencie podczas jego pracy.

POWSTANIE JĘZYKA MML

Liczba systemów komutacyjnych SPC (ang. Stored Program Control) szybko wzrastała już w pierwszej dekadzie ich istnienia, tj. w latach 1960—1970. Wraz z rozwojem tych systemów pojawiły się nie znane dotychczas problemy programowania, które zwróciły uwagę Komitetu CCITT. Dążąc do ich rozpoznania rozpoczęto studia nad zagadnieniami oprogramowania w różnych fazach jego tworzenia. Zauważono już wówczas potrzebę komunikacji z systemem, zarówno w fazie jego powstawania (testowanie systemu i jego oprogramowania), jak i podczas działania (zarządzanie i utrzymanie). W 1972 roku powzięto decyzję opracowania języka komend dla celów utrzymania i zarządzania. Za pomocą ankiet rozsyłanych w latach 1973—1976 do różnych firm telekomunikacyjnych na świecie zestawiono zbiór wymagań dotyczących języka komunikacji z systemem SPC i określono zakres jego zastosowania. Pierwsza wersja języka, powstała w 1976 roku, zawierała opis języka wejściowego i podstawowych procedur dialogu. W następnych latach prowadzono prace nad rozszerzeniem definicji języka i opracowaniem języka wyjściowego. Zaowocowały one utworzeniem obecnej wersji języka opublikowanej w 1985 roku, w tzw. Czerwonej Księdze CCITT [1].

ZASTOSOWANIE

Język MML umożliwia tworzenie sprzężenia operatora z systemem, służącego do wspomagania zadań związanych z eksploatacją systemu, tj. jego zarządzaniem i utrzymaniem oraz zadań związanych z instalowaniem nowego systemu lub zmianą wyposażenia systemu pracującego. Funkcje związane z zarządzaniem obejmują nie tylko zadania administracyjne, takie jak tworzenie statystyk ruchowych, operowanie zasobami danych dotyczących abonenta i trafikacją połączeń, lecz również zadania związane ze sterowaniem systemu, jak np. zmiany strategii kierowania ruchem czy kontrola konfiguracji systemu. W zakres funkcji utrzymaniowych wchodzą m.in.: zadania uruchamiania testów lokalizujących uszkodzenia, nadzorowania pracy podsystemów, kontrolowania obciążenia systemu sterującego, zmiany jego konfiguracji i pielęgnacji oprogramowania.

Uwzględniając potrzeby komunikacji, występujące już podczas instalowania systemu, dostosowano MML do zadań powiązanych z całym zakresem uruchamiania sprzętu i oprogramowania. Przewidziano również możliwość uruchamiania zestawu testów dopuszczających instalowany system do eksploatacji (kontrola zgodności ze specyfikacją systemu).

Terminale MML mogą być instalowane przy systemie lub wyniesione poza system. W tym drugim wypadku mogą obsługiwać jeden lub więcej systemów połączonych w sieć. Jest to niezwykle istotne, gdyż pozwala na centralizację funkcji MML przez tworzenie centrów eksploatacyjno-utrzymaniowych.

KOMUNIKACJA CZŁOWIEK — MASZYNA

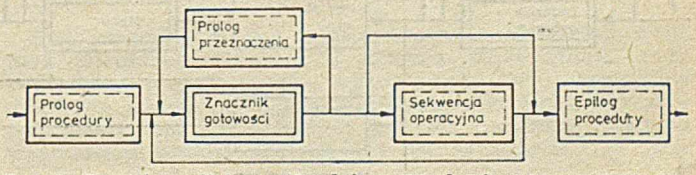
Na komunikację człowieka z maszyną składa się przekazywanie informacji do systemu, tj. jej wprowadzanie (ang. inputs) oraz przekazywanie jej z systemu, tj. wyprowadzanie (ang. outputs). Sekwencja wprowadzania informacji do systemu i wyprowadzania odpowiedzi z systemu (ang. response outputs) tworzy dialog (ang dialogue) operatora z systemem. Dialog odbywa się według ściśle określonych reguł następstwa składających się na procedurę dialogu (ang. dialogue procedure). Niezależnie od dialogu może występować z inicjatywy systemu tzw. wyprowadzanie pozadialogowe (ang. outputs outside dialogue). Dialog i wyprowadzanie pozadialogowe następują sekwencyjnie, lecz bez określonych reguł następstwa. Oprócz aspektów proceduralnych komunikacji z maszyną, każdy element przekazywanych informacji musi być zapisany zgodnie z przyjętymi regułami syntaktycznymi. Syntaktyka wprowadzania jest opisana językiem wejściowym (ang. input language), a wyprowadzana — językiem wyjściowym (ang. output language).

Informacje do systemu są przekazywane w postaci komend i dyrektyw. Komendy (ang. command) przenoszą informacje sterujące do systemu, natomiast dyrektywy (ang. directive) służą do usprawnienia komunikacji z systemem. Przy użyciu dyrektyw można żądać od systemu dodatkowych informacji pomocniczych, na przykład, w wyborze komendy lub wprowadzania parametrów.

PROCEDURA DIALOGU

Procedura dialogu rozpoczyna się i kończy z inicjatywy operatora. Otwierana jest ona przez **prolog procedury** (ang. procedure prologue), po którym może wystąpić **prolog przeznaczenia** (ang. destination prologue), a następująca po nim **interakcyjna sekwencja operacyjna** (ang. interactive operating sequence) jest zakończona **epilogiem procedury** (ang. procedure epilogue) (rys. 1).

Prolog procedury składa się z czynności przygotowawczych przed dialogiem związanych z uaktywnieniem terminala i przedstawieniem się systemu. Może on zawierać opcjonalną procedurę identyfikacji użytkownika i jego autoryzacji, ograniczenia dostępu do terminala lub pewnych grup



Rys. 1. Procedura dialogu. (Zgodnie z zasadami opracowanego metajęzyka opisu syntaktyki MML pojedynczy symbol prostokątny oznacza niedostatecznie zdefiniowany element wejściowy, podwójny — element wyjściowy, a symbol z wewnętrzną linią przerywaną — złożony element wejściowo-wyjściowy)

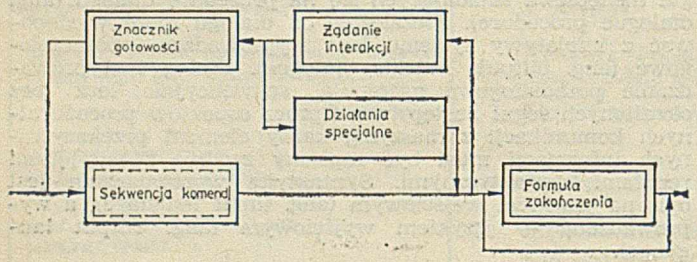
komend. Prolog występuje tylko raz podczas dialogu i kończy się wydrukowaniem znacznika gotowości, po którym może nastąpić prolog przeznaczenia lub bezpośrednio interakcyjna sekwencja operacyjna, którą można nazwać krótko — sekwencją operacyjną.

W prologu przeznaczenia wskazywane jest urządzenie, do którego odnosić się będą komendy występujące w dialogu, przez podanie identyfikatora systemu lub numeru procesora. Następuje po tym wydruk nagłówka tego systemu dla zaznaczenia, że wskazany odbiorca komend jest osiągalny, bądź — komunikat o odmowie lub niemożliwości skomunikowania się z odbiorcą. Dopuszczalną opcją zastępującą prolog przeznaczenia jest podanie identyfikatora odbiorcy bezpośrednio w bloku parametrów komendy.

Epilog procedury stosowany jest na zakończenie dialogu i zawiera szereg czynności związanych z deaktywacją terminala.

SEKWENCJA OPERACYJNA

Pomiędzy prologiem i epilogiem znajduje się właściwa część dialogu — sekwencja operacyjna, mająca postać interakcji, służących wprowadzaniu jednej lub kilku komend do systemu. Każda z nich jest wprowadzana w postaci sekwencji komendy (ang. command sequence) określonej syntaktyką języka wejściowego. Po zakończeniu sekwencji komendy, ze strony systemu może być zgłoszone żądanie interakcji w postaci kolejnych komend operatora lub działań specjalnych, na przykład czynności manualnych obsługi. Następujące po sobie komendy i działania specjalne mogą wystąpić wtedy, gdy wskutek częściowego wykonania zadanych w komendzie funkcji, system domaga się dostarczenia dodatkowych informacji, wymagających oceny i decyzji człowieka (rys. 2).



Rys. 2. Sekwencja operacyjna

SEKWENCJA KOMENDY

Tworząc język MML przewidziano jego implementację z wprowadzaniem komend, przy użyciu jednej z poniższych metod lub ich kombinacji (rys. 3):
— wprowadzanie bezpośrednie,
— wybór z menu,
— wypełnianie formularza.

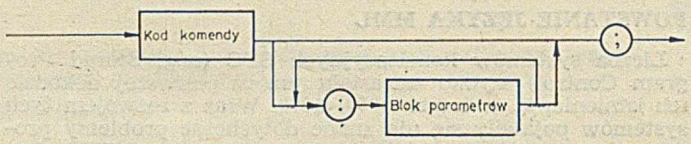
Przydatność poszczególnych metod wprowadzania komend zależy od liczby zadań spełnianych przez język oraz od założonego poziomu wyszkolenia obsługi. Gdy MML spełnia wiele różnych rodzajów zadań, należy stosować metodę wyboru z menu, przy czym wybór może być wielostopniowy — od ogólnego zakresu do konkretnego zadania. W celu ustalenia metody wprowadzania komend operator ma do dyspozycji dyrektywy. Mogą one być implementowane w formie skrótów identyfikatorów menu i formularzy lub po prostu w postaci klawiszy funkcyjnych. W pierwszym wypadku dobrym przykładem identyfikatora formularza może być kod komendy zakończony znakiem zapytania.

W implementacji języka MML można przyjąć alternatywną wersję **samoistnego menu** (ang. spontaneous menu) wyprowadzanego zamiast znacznika gotowości. Jeżeli jest przewidziane bezpośrednie wprowadzanie komendy, to musi ono być także możliwe po samoistnym wyprowadzeniu menu. Niewprawny operator, który nie zna kodu lub struktury komendy, może żądać wydrukowania menu i formularza do wprowadzania parametrów. Wybór z menu, niezależnie od liczby stopni uściśleń zadania, kończy wydruk menu komend. Zakłada się, że operator może żądać pomocy systemu również podczas wyboru komendy i wypełniania formularza parametrów. W takim wypadku procedury selekcji komendy i wypełniania formularza przybierają formę interakcji: zapytanie operatora — wyjaśnienie systemu.

Skompletowanie komendy kończy się potwierdzeniem jej wprowadzenia, w postaci akceptacji lub odmowy. W drugim wypadku, po wskazaniu błędu, może nastąpić procedura jego korekcji.

JĘZYK WEJŚCIOWY

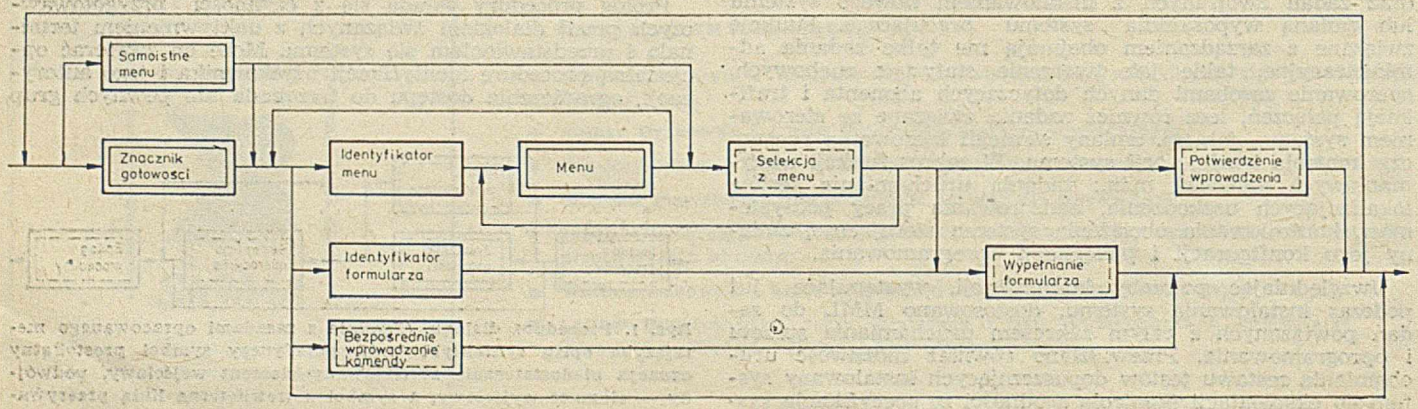
Każdą komendę rozpoczyna **kod komendy** określający funkcję, która ma być wykonana przez system. Gdy wymagane są dalsze informacje, po kodzie komendy następuje blok lub **bloki parametrów** (ang. parameters block) oddzielone separatorami i zakończone znakiem wykonania (rys. 4).



Rys. 4. Struktura komendy (symbole owalne oznaczają ostatecznie zdefiniowane elementy metajęzyka opisu MML)

Kod komendy może zawierać do trzech identyfikatorów rozdzielonych separatorami. Zakłada się, że winny one stopniowo przybliżać znaczenie funkcjonalne komendy, na przykład określać zakres funkcjonalny, rodzaj obiektu, rodzaj działania.

Blok parametrów zawiera informacje niezbędne do wykonania funkcji zapisanej w kodzie komendy i może składać się z różnej liczby parametrów oddzielonych separatorami. Parametry w bloku określane są albo pozycyjnie, albo przez nazwę. **Parametry określone pozycyjnie** (ang. position-defined parameter) muszą być podawane w ustalonej kolejności. Nie dotyczy to **parametrów określonych przez nazwę** (ang. name-defined parameter), których wartość podawana jest wraz z przypisaną im nazwą. Parametry określone pozycyjnie mogą być również poprzedzone nazwą. W obu wypadkach nie muszą być wprowadzane wszystkie parametry — zamiast parametrów pominiętych system podstawia wówczas wartość domyślną (ang. default value).



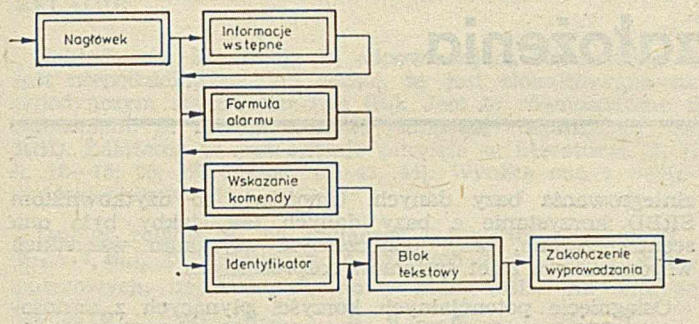
Rys. 3. Sekwencja komendy

W celu usystematyzowania pojęciowego przyjęto, iż wartość parametru składa się z argumentów, złożonych z kolei z jednostek informacji. Argumenty są proste lub złożone, jeżeli zawierają kilka jednostek informacji. W celu usprawnienia wprowadzania parametrów przewidziano możliwość grupowania informacji opisaną syntaktyką języka wejściowego.

JĘZYK WYJŚCIOWY

Przekazanie informacji z systemu może następować jako bezpośrednie wyprowadzenie odpowiedzi na komendę podawaną podczas dialogu lub jako wyprowadzenie pozadialogowe. Ten drugi rodzaj stanowi wyprowadzanie inicjowane przez system w postaci komunikatów — a jego typowym przykładem są alarmy. Należy do niego także niebezpośrednie (ang. delayed) wyprowadzenie odpowiedzi na komendy podawane podczas dialogu, których wykonanie wymaga pewnej zwłoki, na przykład komunikat o parametrach ruchu.

Wyprowadzenie pozadialogowe (rys. 5), tak jak początek dialogu, rozpoczyna się zawsze od nagłówka, podawanego głównie w celach dokumentacyjnych. W nagłówku umieszcza się ogólne informacje nie związane z funkcją komendy, jak: identyfikator urządzenia nadawczego (systemu, procesora), datę i czas, numer komunikatu itp.



Rys. 5. Wyprowadzenie pozadialogowe

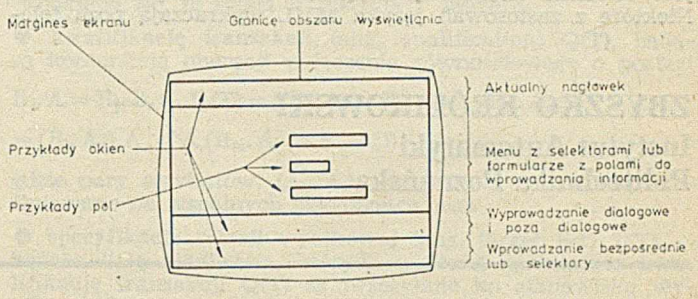
Po nagłówku mogą być opcjonalnie podane wstępne informacje o rodzaju wyprowadzenia (np. utrzymaniowe lub statystyczne) oraz dane identyfikujące odbiorcę. Następująca po tym opcjonalna formuła alarmu (ang. alarm statement), określająca stopień ważności alarmu i jego źródło, może zarazem zawierać zmienny tekst związany z przyczyną komunikatu oraz wskazanie komendy (ang. command reference), która spowodowała jego wyprowadzenie przez podanie jej numeru i ewentualnie tekstu wyjaśniającego. Identyfikator określa jednoznacznie rodzaj wyprowadzania w dopuszczalnym repertuarze systemu, co może służyć odnalezieniu dodatkowych informacji wyjaśniających w opisie implementacji języka MML. Blok tekstowy zawiera ewentualne nazwy parametrów, tabele, teksty zmienne i informacje związane z wyprowadzaniem.

PREZENTACJA INFORMACJI

Niezależnie od opisania komunikacji człowiek — maszyna z leksykalnego punktu widzenia, komisja opracowująca język wydała zalecenia dotyczące sposobu jego wykorzystania, zwłaszcza przy użyciu nowoczesnych terminali. Wskazania dotyczą sposobów organizowania przekazu na podstawie zasad obrazowania informacji i ergonomii. Planowane są prace nad wykorzystaniem elementów graficznych do przekazu informacji.

Język MML może być implementowany dla różnych urządzeń wejścia-wyjścia, od dalekopisów do najnowocześniejszych monitorów ekranowych. Zalecenia nie określają typu urządzenia, a jedynie narzucają warunek, aby było ono zdolne co najmniej do posługiwania się kodem znaków określonych zalecanym podzbiorem alfabetu CCITT nr 5 (zawiera alfabet ASCII).

Obszar wyświetlania informacji na ekranie monitora dzieli się na okna (ang. windows) i pola (ang. fields). Okna mają konkretne przeznaczenie, np. okno nagłówka, okno wejściowe itp. (rys. 6). Pola służą do wprowadzania i wyprowadzania informacji. Pola do wprowadzania informacji są dostępne dla operatora oraz systemu (np. wprowadzanie przez system parametru domyślnego), natomiast pola do wyprowadzania informacji są dostępne tylko dla systemu i strzeżone przed zapisem przez operatora. Pola wyznaczają więc granice dostępu dla operatora. Mogą one znajdować się wewnątrz okna, albo przykrywać całe okno. Rozmiary okien mogą zmieniać się podczas dialogu, jedno okno może nawet zniknąć na korzyść drugiego. Okna mogą również zachodzić na siebie, przesłaniając informację lub tworząc część wspólną informacji. W każdym z nich mogą odbywać się osobne działania, np. przeglądanie tekstu może odbywać się niezależnie od pozostałej części ekranu.



Rys. 6. Przykład rozmieszczania informacji na ekranie monitora

Ze względu na możliwości obrazowania wyświetlanej informacji, komisja opracowująca język zaleca korzystanie z nowoczesnych terminali. Proponuje się między innymi przypisanie atrybutów wizualnych (ang. video attributes) poszczególnym znakom lub ich ciągom, całemu oknu, albo polu lub ich częściom. Wśród atrybutów wyróżnia się przykładowo: kolor, stopień podświetlenia, migotanie, podkreślenie, pogrubienie i pochylenie liter (kursywa). W celu zaznaczenia na ekranie miejsca działania zaleca się stosowanie kursora. Powinien on być sterowany zarówno przez system, jak i przez użytkownika. Do zmiany pozycji kursora powinny być przeznaczone osobne klawisze, ewentualnie — zastosowana jedna ze znanych metod wykorzystujących pióro świetlne, drążek sterowniczy (ang. joystick) itp.

SEMANTYKA JĘZYKA MML

Tworząc język MML nie poprzestano na opracowaniu jego zasad syntaktycznych. Do określenia semantyki tworzonych implementacji języka w zaleceniach CCITT podano szczegółową metodykę specyfikacji styku człowiek — maszyna. Określono proponowany zakres zastosowań języka w telekomunikacji z sugestią tworzenia podzbiorów (lub rozszerzeń) dla konkretnych implementacji. W zalecanej metodzie zastosowano procedurę zstępującego uściślania semantyki MML — od ogólnego określenia zakresu aplikacji zgodnej z wymaganiami użytkownika, przez podział na funkcje i dekompozycję tych funkcji, aż do ścisłej specyfikacji styku człowiek — maszyna. Przewidziano pięć faz uściślania semantyki MML:

- identyfikacja potrzeb,
- identyfikacja funkcji MML,
- identyfikacja struktury informacji,
- specyfikacja styku człowiek — maszyna,
- ocena i weryfikacja faz 2, 3 i 4.

W poszczególnych fazach zaleca się stosowanie metod modelowania i dekompozycji funkcji oraz użycia specjalnie utworzonego metajęzyka opisu struktury informacji. Do opisu procedur wymiany informacji na styku człowiek — maszyna proponuje się zastosowanie języka SDL (ang. Specification and Description Language) — opracowanego przez CCITT języka opisu i specyfikacji [1]).

* * *

Zastosowaniem języka MML zainteresowanych jest wiele firm telekomunikacyjnych z różnych stron świata. Przy tworzeniu implementacji wykorzystują one szczególne własności języka, gdyż jego struktura umożliwia tworzenie podzbiorów dla rozmaitych aplikacji i różnego poziomu wyszkolenia obsługi. Do odmiennych zastosowań lub przy

[1]) SDL, obok Chilla i języka MML, jest jednym z trzech opracowanych przez CCITT języków programowania systemów komutacyjnych.

dostosowaniu dialogu do języka narodowego możliwe jest rozszerzenie języka MML bez naruszenia jego struktury objętej definicją CCITT.

Do 1984 roku, w którym Komitet CCITT zatwierdził tzw. Czerwoną Księgę, zarejestrowano 35 implementacji MML, będących w eksploatacji (na pracujących systemach komutacyjnych), 34 implementacje opracowane lub będące w trakcie opracowywania, a ponadto 15 wypadków planowania prac nad zastosowaniem tego języka. Zainteresowanych bądź prowadzących prace związane z implementacją MML jest 38 firm telekomunikacyjnych z 18 krajów świata. Niektóre z zastosowań języka MML wykraczają poza telekomunikację. We wszystkich wypadkach firmy stwierdzają zgodność implementacji z zaleceniami CCITT, a w 10 — rozszerzenie poza te zalecenia.

LITERATURA

[1] CCITT: Man-Machine Language (MML). Recommendations Z.301—Z.341, Red Book CCITT, Geneva, 1985
[2] Hornbach H.: MML-CCITT Man-Machine Language. IEEE Transactions on Communications, Vol. CO-30, No. 6. June 1982.

Nadesłano: 1986.01.13

ZBYSZKO KRÓLIKOWSKI
Instytut Automatyki
Politechnika Poznańska

Generowanie planów wykonywania transakcji w systemach rozproszonych baz danych (1)

Podstawowe pojęcia i założenia

W ostatnich latach obserwuje się gwałtowny rozwój **rozproszonych systemów** komputerowych, i to zarówno pod względem intensyfikacji prac badawczych, jak i zastosowań. Ogólnie, **przez system rozproszony rozumie się system złożony z pewnej liczby niezależnych, rozproszonych geograficznie stanowisk komputerowych, połączonych liniami transmisyjnymi, które umożliwiają wymianę informacji między tymi stanowiskami.** Przez niezależność stanowiska komputerowego w systemie rozproszonym rozumie się zwykle jego autonomiczne zarządzanie.

Potrzeba tworzenia systemów rozproszonych wynika ze zdecentralizowanego charakteru wielu zastosowań informatyki, w tym m.in.: systemów rezerwacji lotniczej i hotelowej, systemów bankowych i pocztowych, systemów wspomagających zarządzanie przedsiębiorstwami wielozakładowymi, systemów administracji itd. Systemy rozproszone w naturalny sposób odpowiadają strukturze tych przykładowo wymienionych zastosowań, natomiast systemy scentralizowane albo nie mogą być stosowane do takich celów, albo są nieefektywne. Systemy rozproszone, w porównaniu z systemami scentralizowanymi, umożliwiają uzyskanie istotnej poprawy parametrów eksploatacyjnych systemu komputerowego, a w szczególności zapewniają zwiększenie efektywności działania systemu, dostępności danych, niezawodności, otwartości systemu i bezpieczeństwa.

Najbardziej ogólnymi i złożonymi systemami rozproszonymi są systemy rozproszonych baz danych (SRBD). W systemach tych mamy do czynienia z połączeniem dwóch podstawowych aspektów: fizycznego rozproszenia systemu oraz zintegrowania bazy danych. Umożliwia to użytkownikom SRBD korzystanie z bazy danych tak, jakby była ona scentralizowana, przy jednoczesnym uzyskaniu wszystkich wspominanych zalet systemów rozproszonych.

Osiągnięcie potencjalnych korzyści płynących z zastosowania SRBD jest uwarunkowane rozwiązaniem jakościowo nowych, trudnych problemów związanych z zarządzaniem takimi systemami. Jednym z najistotniejszych i najtrudniejszych problemów stojących przed projektantem systemu zarządzania rozproszoną bazą danych (SRBD) jest problem optymalizacji planów wykonywania transakcji. Znaczenie tej problematyki wynika z konieczności zapewnienia odpowiedniej efektywności wykonywania transakcji, decydującej o użyteczności całego systemu. W systemach rozproszonych baz danych, uzyskanie pożądanej efektywności przetwarzania jest szczególnie utrudnione, ze względu na wymaganie zarządzania rozproszonego z jednej strony oraz konieczność transmisji i przetwarzania dużych woluminów danych — z drugiej.

SFORMUŁOWANIE PROBLEMU

Problem optymalizacji planów wykonywania transakcji można sformułować następująco. Dla danej transakcji, wymagającej wykonania wielu operacji na danych zlokalizowanych na różnych stanowiskach komputerowych SRBD, należy określić sekwencję wykonywania tych operacji oraz ich lokalizację, tak aby optymalizować wybrane kryterium oceny działania systemu. Generowanie planów wykonywania transakcji w SRBD było przedmiotem licznych studiów [2, 3, 7, 8, 10, 12—14, 18—20, 23—27, 30—33, 35, 38, 40—42, 44].

Wspólną cechą dotychczas opracowanych algorytmów generowania tych planów jest bardzo uproszczony model kosztów wykonywania transakcji w SRBD. W modelu tym uwzględnia się jedynie koszt transmisji danych. Zupełnie pomija się natomiast koszty przetwarzania danych na stanowiskach komputerowych, w szczególności koszt przygotowania transmisji, wykonywania operacji na danych, dekompozycji transakcji, generowania planów rozproszonego wykonywania transakcji oraz koszty systemowe związane z obsługą synchronizacji transakcji, zabezpieczeniem przed upadkiem systemu itp. Przyjęcie takiego modelu kosztów jest równoznaczne z założeniem, że wąskim gardłem **SRBD** jest transmisja danych.

Odnośnie do modelu transmisji zakłada się zwykle, że:
— koszt transmisji nie zależy od odległości między stanowiskami systemu oraz od obciążenia sieci komunikacyjnej,
— koszt transmisji jest liniową funkcją rozmiaru wolumi-

Dr inż. ZBYSZKO KRÓLIKOWSKI ukończył studia na Wydziale Elektrycznym Politechniki Poznańskiej w 1977 r. Pracuje w Zakładzie Badań Operacyjnych i Systemów Komputerowych Instytutu Automatyki Politechniki Poznańskiej. Zajmuje się problematyką sprawności systemów komputerowych, w szczególności systemów rozproszonych baz danych.

nu przesyłanych danych, tj. $c(x) = ax + b$, gdzie b jest kosztem związanym z przygotowaniem sesji transmisyjnej, a x — rozmiarem woluminu przesłanych danych.

W literaturze dotyczącej optymalizacji wykonywania transakcji w SRBD przyjmuje się następując definicję rozproszonej bazy danych: **rozproszoną bazą danych (RBD) nazywa się czwórkę (ST, DL, DF, Loc)**, gdzie:

$ST = \{ST_1, ..., ST_n\}$	jest zbiorem stanowisk komputerowych,
$DL = \{S_1, S_2, ..., S_m\}$	jest zbiorem danych logicznych, nazywanym **logiczną bazą danych,**
$DF = \{R_1, R_2, ..., R_m\}$	jest zbiorem danych fizycznych, nazywanym **fizyczną bazą danych,**
$Loc : DF \rightarrow ST$	jest funkcją określającą lokalizację danej fizycznej $R_l \in DF$ w zbiorze ST.

Logiczna baza danych **DL** reprezentuje **RBD** z punktu widzenia użytkownika. Stosując pojęcia logicznej bazy danych użytkownik definiuje swoje zadanie, nazywane transakcją. Użytkownik nie jest świadomy faktu fizycznego rozproszenia danych, do których żąda dostępu. Nie rozróżnia więc rozproszonej bazy danych od scentralizowanej. Zwykle przyjmuje się, że logiczna baza danych jest widziana przez użytkownika w postaci relacyjnej.

RELACJE

Zakłada się, że relacja R_l odpowiadająca schematowi S_l jest niepodzielna w tym sensie, że jest zlokalizowana na pojedynczym stanowisku **Loc** (R_l). Jest to równoznaczne z założeniem, że relacja stanowi jednostkę rozprzestrzenia w RBD. Założenie to jest ogólnie przyjęte w literaturze [2, 7, 8, 12—15, 20, 26, 30—33, 41, 42, 44]. Wynika ono z następującej obserwacji:

Dzieląc relację R_l na rozłączne fragmenty (relacje) $R_{l1}, R_{i2}, \ldots, R_{im_i}$ pamiętane na m_i różnych stanowiskach komputerowych, uzyskuje się nowy stan RBD, dla którego $DF' = \{R_1, \ldots, R_{i-1}, R_{i+1}, \ldots, R_n, R_{n+1}, R_{n+2}, \ldots, R_{n+m_i}\}$.

Fragmentowi R_{im_j}, gdzie $1 \leqslant m_j \leqslant m_i$, odpowiada w **DF'** relacja R_{n+m_j}. Zadanie użytkownika żądające dostępu do relacji R_l w bazie danych **DF** można zdefiniować używając pojęć bazy danych **DF'** jako zadanie żądające jednoczesnego dostępu do relacji $R_{n+1}, R_{n+2}, \ldots, R_{n+m_i}$.

Przyjmuje się następujące założenia dotyczące relacji:

- jednorodność rozkładu wartości atrybutów, tzn. wszystkie wartości jakiegokolwiek atrybutu pojawiają się z równą częstotliwością,
- niezależność atrybutów, tzn. prawdopodobieństwo $P(V_1 \& V_2)$, że atrybut ma wartość V_1 i atrybut 2 ma wartość V_2 w tej samej krotce relacji, jest równe iloczynowi prawdopodobieństwa $P_1(V_1)$, że atrybut 1 ma wartość V_1, i prawdopodobieństwa $P_2(V_2)$, że atrybut 2 ma wartość V_2, tzn. $P(V_1 \& V_2) = P_1(V_1) * P_2(V_2)$.

Założenia te były przedmiotem szczegółowej analizy Christodoulakisa (m.in. [16, 17]), który wykazał, że prowadzą one do zbyt pesymistycznych oszacowań kosztów wykonywania transakcji. Zwróćmy jednakże uwagę, że powyższe założenia stanowią podstawę teorii współczynnika selektywności [1]) atrybutów połączeniowych relacji. Teoria ta jest powszechnie wykorzystywana w metodach szacowania rozmiarów relacji wynikowych, po wykonaniu operacji relacyjnych, co z kolei jest podstawowym elementem większości znanych z literatury algorytmów generowania planów wykonywania transakcji w **SRBD**. Tak więc, odrzucenie powyższych założeń, na obecnym etapie rozwoju badań nad problematyką optymalizacji planów wykonywania transakcji w **SRBD**, jest niemożliwe.

TRANSAKCJE

Transakcja (ang. read-only transaction) jest elementarną jednostką interakcji użytkownika z **SRBD**. Koncepcja transakcji jako podstawowej jednostki programowej w **SRBD**, znalazła ogólne uznanie i akceptację wśród projektantów systemów baz danych [34, 37]. Koncepcja ta umożliwia bowiem rozłączne rozpatrywanie problemu dostępu do bazy danych pojedynczego użytkownika oraz problemów: synchronizacji współbieżnego dostępu wielu użytkowników i zagwarantowania odpowiedniego stopnia niezawodności systemu [2, 37].

Poniżej przedstawiamy model transakcji przyjętej w literaturze.

Zakłada się, że każda transakcja T zawiera:

- **kwalifikację transakcji** (ang. qualification) Q(T), będącą koniunkcją operacji połączenia równościowego o postaci

$R_i.A = R_j.B$, tj. $Q(T) = \{(R_i.A_k = R_j.A_l) \wedge (R_k.A_m = (R_l.A_n) \wedge \ldots \wedge (R_m.A_p = R_n.A_r)\}$,

gdzie pary atrybutów: (A_k, A_l), (A_m, A_n), ..., (A_p, A_r) są zdefiniowane na wspólnych dziedzinach oraz

- **specyfikacje relacji wynikowej** (ang. target list) T(T) — będącą listą atrybutów, których wartości spełniające kwalifikację transakcji Q(T) są przesyłane na stanowisko wynikowe. Powszechnie [2, 7, 8, 12, 13, 23, 24, 26, 27, 30, 32, 33, 41—44] przyjmuje się założenie, że dla każdego połączenia $R_k \cdot A_k = R_l \cdot B$ występującego w kwalifikacji Q(t) relacje R_k i R_l znajdują się na różnych stanowiskach komputerowych, tzn. $Loc(R_k) \neq Loc(R_l)$.

Dla dowolnej transakcji T można na podstawie jej kwalifikacji skonstruować tzw. **graf transakcji** G(T). Zbiór wierzchołków grafu G(T) odpowiada zbiorowi relacji ujętych w Q(T), natomiast krawędzie grafu G(T) wskazują operacje połączenia na zbiorze wspólnych atrybutów połączeniowych tych relacji. W zależności od struktury grafu transakcji, wyróżnia się następujące rodzaje transakcji: **drzewowe, łańcuchowe, cykliczne i gwiaździste.**

Transakcje można również sklasyfikować w zależności od liczby atrybutów połączeniowych, występujących w poszczególnych elementach Q(T). Wyróżnia się wówczas: **transakcje proste,** w których każda operacja połączenia w Q(T) jest wykonywana na jednym atrybucie połączeniowym oraz **transakcje złożone,** w których operacje połączenia w Q(T) mogą być wykonywane na więcej niż jednym atrybucie połączeniowym.

Oprócz wymienionych wyżej rodzajów transakcji, używa się również pojęcia **transakcja ogólna,** oznaczającego transakcję, której graf G(T) ma dowolną postać, tj. nie ma żadnych ograniczeń co do liczby atrybutów połączeniowych operacji połączenia ujętych w Q(T).

Każdej transakcji T odpowiada niepusty zbiór możliwych algorytmów jej wykonania w SRBD, zwanych **planami wykonywania transakcji.** Plany wykonywania transakcji różnią się między sobą lokalizacją i kolejnością operacji wykonywanych na poszczególnych stanowiskach komputerowych oraz kierunkiem i kolejnością przesyłania danych. Wybór określonego planu wykonywania transakcji jest dokonywany przez procedury optymalizacyjne **SZRBD**, na podstawie przyjętych kryteriów oceny działania **SRBD.**

KRYTERIA OCENY DZIAŁANIA SRBD

Przyjęto dwa kryteria oceny działania **SRBD:** czas odpowiedzi oraz sumaryczne obciążenie systemu. **Czas odpowiedzi** (ang. response time) — jest sumą kosztów od momentu startu pierwszej operacji wykonywanej w ramach realizacji transakcji do momentu zakończenia ostatniej operacji transakcji. W wypadku równoległego wykonywania pewnych operacji (np. równoległa transmisja danych), do kosztów czasu odpowiedzi wlicza się koszt tej spośród operacji równoległych, która jest „najdroższa". **Sumaryczne obciążenie systemu** (ang. total time) — jest sumą kosztów wszystkich operacji realizowanych w ramach wykonywania transakcji — współbieżność operacji nie jest uwzględniana.

Kryterium czasu odpowiedzi reprezentuje punkt widzenia użytkownika, a **kryterium sumarycznego obciążenia systemu** — punkt widzenia administratora SZRBD.

W definicjach obu tych kryteriów używa się pojęcia kosztu, przy czym zakłada się, że koszt może być wyrażony w jednostkach czasu lub liczbie instrukcji kodu maszynowego oprogramowania **SZRBD.** Tę ostatnią jednostkę, można łat-

---

1) Współczynnik selektywności [oznaczany SF $(R_i \cdot A_j)$, gdzie A_j jest atrybutem relacji R_i] jest stosunkiem rozmiaru atrybutu bez duplikatów do rozmiaru dziedziny tego atrybutu.

wo przeliczyć na jednostki czasu, a więc pojęcia kosztu i czasu są ściśle związane. Nie bierze się zatem pod uwagę żadnych innych aspektów oceny działania **SRBD**, na przykład, kosztów finansowych ponoszonych przez użytkowników systemu.

Podstawowy schemat wykonywania transakcji w **SRBD** ma następującą postać [2, 7, 8, 12, 13, 26, 29, 30, 32, 33, 41—43]:

- **wykonaj operacje, które wymagają tylko danych lokalnych** (tzw. operacje unarne),
- **wykonaj wstępne przetwarzanie, w celu redukcji relacji przetwarzanych w następnym kroku,**
- **wykonaj operacje, które wymagają danych z więcej niż jednego stanowiska komputerowego** (operacje binarne),
- **wykonaj operacje końcowe na stanowisku wynikowym, zgodnie ze specyfikacją relacji wynikowej, w celu odpowiedniej prezentacji wyniku.**

Wykazano [14, 36, 39], że powyższy schemat, zgodnie z którym w pierwszej kolejności wykonuje się operacje **unarne**, a następnie operacje **binarne**, minimalizuje koszty wykonywania transakcji. Wykazano ponadto [4, 8, 9, 11, 14, 28, 29, 32, 39], że koszt wykonywania transakcji zasadniczo jest zdeterminowany przez koszt operacji binarnych, a w szczególności operacji połączenia.

OPERACJA PÓŁPOŁĄCZENIA

W środowisku **SRBD** rozwiązaniem alternatywnym w stosunku do operacji połączenia jest operacja półpołączenia (ang. semi-join). **Operacja półpołączenia relacji R_i z relacją R_k** (oznaczana $R_i \rhd\!< R_k$) **jest to połączenie relacji R_i z wynikiem operacji projekcji relacji R_k, na wspólnych atrybutach połączeniowych obu relacji, tzn.**

$$R_i \rhd\!< R_k = \{r_i \in R_i \mid (\exists_{r_k} \in R_k)(r_i[R_i \cap R_k] = r_k[R_i \cap R_k])\}$$

gdzie r_i, r_k są krotkami relacji R_i, R_k.

Jak wynika z powyższej definicji, **operacja półpołączenia, w przeciwieństwie do operacji połączenia jest asymetryczna**, tzn. $R_i \overset{\rhd\!<}{\longrightarrow} R_k \neq R_k \overset{\rhd\!<}{\longrightarrow} R_i$. Operacja półpołączenia $R_i \rhd\!< R_k$ jest kompozycją operacji relacyjnych: projekcji i połączenia. Po wykonaniu operacji projekcji relacji R_k według atrybutu połączeniowego, jej wynik, tzn. wszystkie unikalne (bez duplikatów) wartości atrybutu połączeniowego, są łączone z relacją R_i. Rozmiar relacji R_i zostaje w ten sposób zredukowany tylko do tych krotek, które wystąpią w relacji będącej wynikiem połączenia relacji R_i i R_k.

Operacja półpołączenia, jako alternatywna w stosunku do operacji połączenia, jest w **SRBD** preferowana z dwóch powodów [7]:

- operacja ta ma silne właściwości redukcyjne ($R_i \rhd\!< R_k \subseteq R_i$). Dla kontrastu, operacja połączenia ($R_i \rhd\!\lhd R_k$) może powodować wzrost rozmiaru bazy danych; w najgorszym wypadku: rozmiar ($R_i \rhd\!\lhd R_k$) = rozmiar (R_i) * rozmiar (R_k),
- operacja półpołączenia może być wykonywana przy mniejszych kosztach transmisji danych niż operacja połączenia. Do wykonania operacji półpołączenia $R_i \rhd\!\lhd R_k$ jest konieczna jedynie transmisja wyniku operacji projekcji na relacji R_k, natomiast do wykonania operacji połączenia $R_i \rhd\!\lhd R_k$ jest konieczna transmisja całej relacji R_i lub R_k.

Oczywiście operacja półpołączenia może w niektórych sytuacjach dawać mniejszy efekt niż operacja połączenia, ponieważ operacja $R_i \rhd\!< R_k$ redukuje tylko relację R_i, a operacja $R_i \rhd\!\lhd R_k$ może redukować obie relacje R_i i R_k. Jednakże, w takim wypadku pojedyncza operacja połączenia może być zastąpiona dwiema operacjami półpołączenia, zazwyczaj przy koszcie mniejszym niż koszt operacji połączenia [7]. Poniżej przedstawiono przykład wykorzystania sekwencji operacji półpołączenia do realizacji transakcji wymagającej wykonania operacji połączenia [21].

Przykład

Rozważamy rozproszoną bazę danych zawierającą następujące schematy relacji:
CZĘŚCI (Część, Cena)
POWIĄZANIA (Producent, Część, Odbiorca)
ODBIORCA (Odbiorca, Miasto)
Poszczególne relacje są rozproszone na stanowiskach komputerowych SRBD w następujący sposób: **Loc (CZĘŚCI)** = ST_1, **Loc (ODBIORCY)** = ST_2, **Loc (POWIĄZANIA)** = ST_3. Stan bazy danych przedstawiono w tabelach 1—3.

Tabela 1. Relacja CZĘŚCI (szerokość krotki = 30 bajtów)

Część	Cena
Minikomputer	1 500
Mikrokomputer	150
Komputer	10 000
20 bajtów	10 bajtów

Tabela 2. Relacja ODBIORCY (szerokość krotki = 50 bajtów)

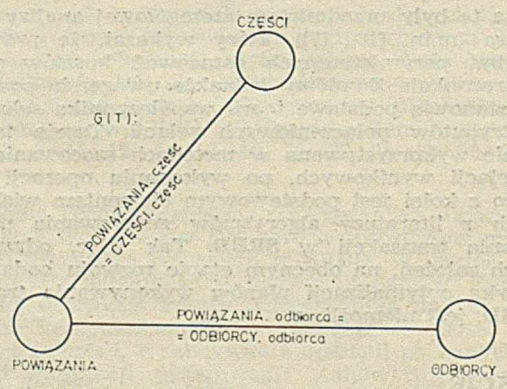
Odbiorca	Miasto
Politechnika Poznańska	Poznań
Uniwersytet Poznański	Poznań
Uniwersytet Warszawski	Warszawa
30 bajtów	20 bajtów

Tabela 3. Relacja POWIĄZANIA (szerokość krotki = 60 bajtów)

Producent	Część	Odbiorca
DEC	Minikomputer	Politechnika Poznańska
IBM	Minikomputer	Politechnika Poznańska
HP	Komputer osobisty	Uniwersytet Warszawski
HP	Komputer osobisty	Uniwersytet Poznański
Sharp	Minikomputer	Politechnika Warszawska
Zilog	Mikrokomputer	Politechnika Gdańska
Cray	Komputer	Uniwersytet Warszawski
10 bajtów	20 bajtów	30 bajtów

Transakcja T:

Q(T) = {(POWIĄZANIA. Część = CZĘŚCI. Część) ∧ (POWIĄZANIA.Odbiorca = ODBIORCY.Odbiorca)}
T(T) = {Część, Cena, Producent, Odbiorca, Miasto}
jest transakcją drzewową (rys. 1). Jako alternatywę w stosunku do dwóch operacji połączenia z Q(t) (rys. 1).



Przykładowa transakcja drzewowa

Rozważmy następującą sekwencję operacji półpołączenia: POWIĄZANIA ▷< CZĘŚCI, POWIĄZANIA ▷< ODBIORCY, CZĘŚCI ▷< POWIĄZANIA, ODBIORCY ▷< POWIĄZANIA.

Wykonując operację POWIĄZANIA ▷< CZĘŚCI redukujemy relację POWIĄZANIA przez eliminację krotek, w

których Część = „komputer osobisty", ponieważ taka wartość atrybutu Część w relacji CZĘŚCI nie występuje. Wykonując operację POWIĄZANIA ▷< ODBIORCY redukujemy relację POWIĄZANIA przez eliminację krotek, w których Odbiorca = „Politechnika Warszawska" i Odbiorca = „Politechnika Gdańska".

Tak więc po wykonaniu operacji półpołączenia: POWIĄZANIA CZĘŚCI i POWIĄZANIA ▷< ODBIORCY otrzymujemy nowy stan relacji POWIĄZANIA (tab. 4).

Tabela 4. Relacja POWIĄZANIA'

Producent	Część	Odbiorca
DEC	Minikomputer	Politechnika Poznańska
IBM	Minikomputer	Politechnika Poznańska
Cray	Komputer	Uniwersytet Warszawski

Wykonując następnie operacje półpołączenia: CZĘŚCI ▷< POWIĄZANIA oraz ODBIORCY ▷< POWIĄZANIA otrzymujemy zredukowane relacje: CZĘŚCI' i ODBIORCY' (tab. 5 i 6).

Tabela 5. Relacja CZĘŚCI'

Część	Cena
Minikomputer	1 500
Komputer	10 000

Tabela 6. Relacja ODBIORCY'

Odbiorca	Miasto
Politechnika Poznańska	Poznań
Uniwersytet Warszawski	Warszawa

Otrzymaliśmy w ten sposób nowy stan bazy danych z relacjami CZĘŚCI', POWIĄZANIA' i ODBIORCY'. Wykonanie założonej w transakcji T sekwencji operacji połączenia na relacjach POWIĄZANIA', CZĘŚCI', ODBIORCY' daje wynik identyczny jak z połączenia relacji POWIĄZANIA, CZĘŚCI, ODBIORCY.

Jak widać z przedstawionego przykładu, wykonanie odpowiedniej sekwencji operacji półpołączenia redukuje przetwarzane relacje przez wyeliminowanie krotek, które nie znajdą się w relacji wynikowej.

SKUTECZNOŚĆ OPERACJI PÓŁPOŁĄCZENIA

Na zakończenie omawiania operacji półpołączenia, warto przedstawić w skrócie podstawowe wyniki uzyskane w badaniach nad efektywnością tej operacji, dla wybranych klas transakcji.

W wielu pracach [4—6] przeanalizowano dokładnie możliwości redukcyjne operacji półpołączenia dla transakcji drzewowych i cyklicznych. Wnioski wynikające z tych badań, opublikowane w pracy z tego zakresu [4], można zreasumować w następujący sposób: wykorzystując sekwencję operacji półpołączenia do realizacji transakcji drzewowej, można osiągnąć stan pełnej redukcji relacji, na których jest wykonywana ta transakcja. Stan pełnej redukcji rozumie się intuicyjnie jako ograniczenie rozmiarów łączonych relacji do tych tylko krotek, które są niezbędne dla realizacji operacji połączenia. Inaczej mówiąc wszystkie te krotki, które nie znajdą się w relacji wynikowej, są eliminowane przez operację półpołączenia. Warto zaznaczyć, że powyższy wniosek dotyczy zarówno transakcji prostych, jak i złożonych [5].

Dla klasy transakcji cyklicznych problem redukcji rozmiarów relacji przez sekwencję półpołączenia jest znacznie trudniejszy i należy do klasy problemów silnie NP-zupełnych [4, 5, 22]. Otrzymanie stanu pełnej redukcji dla transakcji cyklicznych, o ile w ogóle jest możliwe, wiąże się z poniesieniem znacznie większych kosztów niż w wypadku transakcji drzewowych.

W konsekwencji wspomnianego wcześniej założenia o dominacji kosztów transmisji w ogólnych kosztach wykonywania transakcji w SRBD, zdecydowana większość opracowanych dotychczas algorytmów generowania planów wykonywania transakcji, koncentruje się wyłącznie na redukcji woluminów danych przesyłanych pomiędzy stanowiskami SRBD i wykorzystuje w tym celu strategię operacji półpołączenia.

* * *

W części drugiej artykułu dokonamy przeglądu opracowanych do tej pory algorytmów generowania planów wykonywania transakcji w systemach rozproszonych baz danych.

LITERATURA

[1] Allchin J. E., McKendry M. S.: Synchronization and recovery of actions. Proc. 2nd Annual ACM Symp. on Principles of Distributed Computing, pp. 31—44, 1983

[2] Apers P. M., Hevner A. R., Yao S. B.: Optimization algorithms for distributed queries. IEEE Trans. on Software Engineering, Vol. SE-9, No. 1, pp. 57—67, January 1983

[3] Astrahan M., Kim W., Schkolnick M.: Evaluation of the system R access path selection mechanism. Research Report RJ 2798, IBM Research Laboratory, San Jose, 1980

[4] Bernstein P. A., Chiu D. W.: Using semi-joins to solve relational queries. Journal of the ACM, Vol. 28, No. 1, pp. 25—40, 1981

[5] Bernstein P. A., Goodman N.: The power of natural semijoins. SIAM J. Comput., Vol. 10, No. 4, pp. 751—771, 1981

[6] Bernstein P. A., Goodman N.: The power of inequality semi-joins, Information Systems, Vol. 6, No. 4, pp. 225—265, 1981

[7] Bernstein P. A. i inni: Query processing in a system for distributed databases (SDD-1). ACM Trans. on Database Systems, Vol. 6, No. 4, pp. 602—625, 1981

[8] Black P. A., Luk W. S.: A new heuristic for generating semi-joins programs for distributed query processing, Proc. Conf. IEEE COMPSAC, pp. 581—588, 1981

[9] Bitton D., DeWitt D. J., Turbyfill C.: Benchmarking database systems a systematic approach. Computer Sciences Technical Report No. 526, University of Wisconsin-Madison, December 1983

[10] Cellary W., Meyer D.: A multi-query approach to distributed processing in a relational distributed data base management system. Proc. Intern. Symposium on Distributed Data Bases, C. Delobel, W. Litwin, ads. pp. 99—121, Paris, 1980

[11] Ceri S., Pelagatti G.: Distributed Databases — Principles and Systems, McGraw-Hill, New York, 1984

[12] Chen A. L. P., Li V. O. K.: Optimizing Star queries in distributed database system. Proc. 10th Intern. Conf. on VLDB, pp. 344—347, 1984

[13] Chiu Dah-Ming, Bernstein P. A., Ho Yu-Chi: Optimizing chain queries in distributed database systems. SIAM J. Comput., Vol. 13, No. 1, pp. 116—134, 1984

[14] Chu W. W., Hurley P.: Optimal query processing for distributed database systems. IEEE Trans. on Computers. Vol. C-31, No. 9, pp. 835—850, 1982

[15] Chung Ch. W., Irani K. G.: A semi-join strategy for distributed query optimization. Proc. 4th Intern. Conf. on Distributed Computing, pp. 368—377, San Francisco, 1984

[16] Christodoulakis S.: Estimating block selectivities. Information Systems, Vol. 9, No. 1, pp. 69—81, 1984

[17] Christodoulakis S.: Implication of certain assumptions in database performance evaluation. ACM Trans. on Database Systems, Vol. 9, No. 2, pp. 163—186, 1984

[18] Daniels D.: Query complication in a distributed database system. Research Report RJ 3423, IBM Research Laboratories, San Jose, 1982

[19] Dayal U., Goodman N.: Query optimization for Codasyl Database Systems. Proc. ACM-SIGMOD Conf. on Management of Data, pp. 138—150, Orlando, 1982

[20] Epstein R., Stonebraker M. R., Wong E.: Distributed query processing in a relational data base system. Proc. ACM-SIGMOD Intern. Conf. on Management of Data. pp. 169—180, Austin, 1978

[21] Goodman M., Shmueli O.: Tree queries: a simple class of relational queries. ACM Trans. on Database Systems, Vol. 7, pp. 653—677, 1982

[22] Goodman N., Shmueli O.: The tree property is fundamental for query processing, Proc. ACM-SIGACT-SIGMOD Conf. on Principles of Database Systems. pp. 40—48, Los Angeles, March 1982

[23] Goldhirsch D., Yedwab L.: Processing read only queries — over views with generalization, Proc. 10th Intern. Conf. on VLDB, pp. 344—347, 1984

[24] Gouda M. G., Dayal U.: Optimal semi-join schedules for query processing in local distributed database systems. Proc. ACM-

-SIGMOD. Intern. Conf. on Management of Data, Ann Arbor, pp. 164—175, 1981
[25] Griffiths et al.: Access path selection in a relational database management systems. Research Report RJ 2429, IBM Research Laboratories, San Jose, 1979
[26] Hevner A. R., Yao S. B.: Query processing in distributed database systems. IEEE Trans. on Software Engineering, Vol. SE-5, No. 5, pp. 177—187, 1979
[27] Huang T. K.: A cost evaluation model for processing distributed query. Proc. 18th Hawaii International Conf. on System Sciences, 1985
[28] Jarke M., Koch J., Schmidt J. W.: Introduction to query processing. Working Paper CRIS-73-GBA 84-48/CR, Graduate School of Business Administration, New York University, March 1984
[29] Jarke M., Koch J.: Query optimization in database systems. Computing Surveys, Vol. 16, No. 2, pp. 111—152, 1984
[30] Kambayashi Y. i inni: Query processing for distributed database using generalized semi-joins. Proc. ACM-SIGMOD Intern. Conf. on Management of Data, pp. 151—160, Orlando, 1982
[31] Kerschberg L., Ting P. D., Yao S. B.: Query optimization in star computer networks. ACM Trans. on Database Systems, Vol. 7, No. 4, pp. 678—711, 1982
[32] Luk W. S., Luk L.: Optimizing semi-join programs for distributed query processing. Proc. Conf. ICOD-2, pp. 298—316, Cambridge, 1983
[33] Perrizo W.: A method for processing distributed database queries. IEEE on Software Engineering, Vol. SE-10, No. 4, pp. 466—470, 1984
[34] Schwarz P. M., Spector A. Z.: Synchronizing Shared Abstract Types, IEEE Trans. on Computer Systems, Vol. 2, No. 3, 1984
[35] Selinger P. G., Adiba M. E.: Access path selection in distributed database management systems. Conf. ICOD, Aberdeen, Scotland, 1980. S. M. Deen and P. Hammersley, eds.
[36] Smith J. M., Chang P. Y.: Optimizing the performance of a relational algebra database interface. Communications of the ACM, Vol. 18, No. 10, pp. 568—579, 1975
[37] Spector A. Z., Schwarz P. M.: Transaction — a construct for reliable distributed computing. Operating Systems Review, Vol. 17, No. 2, pp. 18—35, 1983
[38] Toan N. G.: Distributed query management for a local network database systems. Proc. 2nd Intern. Conf. on Distributed Computing Systems, Paris, 1981
[39] Ullman J.: Principles of database systems. Computer Science Press, 1980
[40] Vard Y. L., Vrbsky V.: Distributed query pprocessing allowing for redundant data. Proc. 4th Intern. Conf. on Distributed Computing, pp. 389—396, 1984
[41] Wong E.: Retrieving dispersed data from SDD-1 — a system for distributed data bases. Proc. 2nd Berkeley Conf. on Distributed Data Management and Computer Networks, pp. 217—235, 1977
[42] Yoshikawa M., Kambayashi Y.: Processing inequality queries based on generalized semi-join, Proc. 10th Intern. Conf. on VLDB, pp. 416—428, 1984
[43] Yu C. T., Lam K., Ozsoyoglu M.: Distributed query optimization for tree queries. Technical Report. Dep. of Information Engineering, University of Illinois at Chicago Circle, pp. 1—55, 1980
[44] Yu C. T., Chang C. C.: On the design of a query processing strategy in a distributed database environment. Proc. SIGMOD 83 Conf., pp. 30—39, New York, 1983.

Nadesłano: 1986.02.04

Ogłoszenia • Ogłoszenia • Ogłoszenia • Ogłoszenia

BIURO USŁUG KOMPUTEROWYCH. Pośrednictwo sprzedaży mikrokomputerów, części zamiennych. Warszawa, tel. 41-44-48.

EO/272/K/86

Ogłoszenia • Ogłoszenia • Ogłoszenia • Ogłoszenia

ODiTK
Gdańsk

mikrokomputery

doradztwo
ODiTK
może poprawić
trafność decyzji

EO/524/K/86

ZDZISŁAW PŁOSKI
Wrocław

Współpraca LOGO Sinclaira ze stacją dyskietek

O tym, jak możliwość wykorzystania dyskietek podnosi rangę mikrokomputera, nie trzeba nikogo przekonywać. Popularny ZX-Spectrum najczęściej występuje u nas w konfiguracji z magnetofonem. Nietypowe pamięci kasetowe Sinclaira o nazwie Microdrive są przy polskich Spectrach rzadkością. Co więcej, 80 KB pojemności jednej kasety do Microdrive, to nie jest jednak 600 KB dyskietki.

Ostatnio otworzyła się przed Spectrum perspektywa współpracy z dyskietkami. Przedsiębiorstwo Apina dostarcza mechanizm (niestety tylko jeden w zestawie) napędu ośmiocalowej dyskietki oraz rozszerzenie oprogramowania systemowego Spectrum, o dźwięcznej nazwie DOS (dokładnie: AC-DOS ver. 10), organizujące kartotekę plików na dyskietce. Wprawdzie za te udogodnienia tanio nie liczy, ale to jest zupełnie inna sprawa.

Stając się więc szczęśliwym posiadaczem konfiguracji Spectrum z dyskietką należy spieszyć się (bo — niestety — jeden z trzech testowanych przez autora egzemplarzy napędów zaczął niedomagać już po 5 godzinach pracy), aby zrobić z jej pomocą coś pożytecznego. Sam system AC-DOS pozwala składować (i wprowadzać do pamięci operacyjnej) programy w Basicu i dowolne bloki bajtów (może z wyjątkiem załączonego na niezabezpieczonej przed przypadkowym skasowaniem dyskietce programu formatującego), oglądać katalog plików dyskietki na ekranie, przemianowywać i usuwać pliki (jeśli w tym czasie nie zapisał się jakiś plik o niepoprawnej nazwie) oraz wykonywać kilka innych, pożytecznych czynności. Pozwala również umieszczać w programie napisanym w Basicu dowolne z jedenastu poleceń.

Z kolei w języku Logo Sinclaira — przystosowanym do współpracy z pamięcią Microdrive — nie założono, że Spectrum otrzyma w przyszłości (tu: przy udziale Apiny) standardowy i wygodny typ pamięci zewnętrznej. Powstał więc dość interesujący problem, jak możliwie szybko i bezboleśnie uzyskać w Logo kontakt z dyskietką.

Autor artykułu na krótko przed otrzymaniem dyskietek postanowił uzyskać „ploskie", czyli własne Logo. Sięgając do obrazu pamięci interpretera (w wersji 1.6 Logo SOLI/LCSI — Les Systems d'Ordinateur Logo International oraz Logo Computer Systems Inc.) uzyskuje się następujące informacje: tablice sygnałów — powitalnego i zgłoszenia edytora, które rezydują odpowiednio od adresów 40230 i 33096 oraz tablice większości błędów (oznajmianych po polsku — bo na ploskie było przerabiane akurat Logo polskie) — od adresu 40280. Oprócz tego od adresu 50402 rozciąga się obszar 2000 bajtów wypełnionych zerami. Obszar ten jest przeznaczony dla edytora Logo, który umieszcza tam teksty wszystkich procedur aktualnie zgłoszonych do obróbki.

Dzięki odkryciu tych informacji, koncepcja szybkiego, choć prowizorycznego połączenia Logo z dyskietką powstała prawie natychmiast. Koncepcję tę podano poniżej w postaci opisowego algorytmu.

Sprzęg między Logo Sinclaira a dyskietkami obsługiwanymi przez DOS Apiny

Aby uzyskać możliwość składowania i odzyskiwania z dyskietki procedur Logo, należy wykonać następujące działania:

I. Składowanie na dyskietce (wykorzystując obszar edytora Logo)

1. Wprowadź w obszar roboczy edytora procedury potrzebne do składowania (instrukcją **edit** systemu programowania Logo)
2. Wyjdź z Logo do współpracującego z nim programu w Basicu (za pomocą instrukcji **bye**)
3. Wywołaj z Basica system AC-DOS instrukcją

RANDOMIZE USR 15360

4. Zeskładuj 2000 bajtów obszaru roboczego edytora, który oprócz zer zawiera obecnie potrzebne procedury. W tym celu wydaj systemowi AC-DOS polecenie:

SAVE „nazwa-pliku-na-dyskietce" CODE 52402, 2000

5. Wróć z systemu DOS do Basica, wydając polecenie

RETURN

6. Wróć z Basica do Logo (instrukcja RUN).

II. Odzyskiwanie procedur z dyskietki (wykorzystując również obszar edytora Logo)

1. Upewnij się, że w obszarze roboczym edytora nie ma nic, czego straty mógłbyś żałować (instrukcją **edit** bez parametru)
2. Wyzeruj obszar edytora Logo instrukcją **edit**
3. Wykonaj czynności opisane w punktach I.2 i I.3
4. Sprowadź do pamięci operacyjnej w obszar roboczy edytora Logo plik z procedurami, utworzony uprzednio na dyskietce
5. Wykonaj czynności opisane w punktach I.5 i I.6
6. Wywołaj edytor bez zmiany jego obszaru roboczego (instrukcją **edit** bez parametrów)
7. Zaakceptuj (albo nie zaakceptuj) to, co zostanie wyświetlone (przyciskając jednocześnie klawisze CAPS-SHIFT + SYMBOL-SHIFT, a następnie C).

Powyższa metoda, która okazała się w pełni skuteczna, ma trzy nieistotne wady:
— bardzo niewygodne są liczne skoki po poziomach Logo-Basic-DOS i z powrotem,
— bardzo nieoszczędne jest składowanie procedur wraz z pustą częścią obszaru,
— każdorazowo używa się obszaru roboczego edytora Logo, co powoduje niebezpieczeństwo niezamierzonego skasowania wprowadzonych właśnie poprawek. Wada ta jest jednak w części zaletą, gdyż zawsze można jeszcze zobaczyć to, co się na dysk wysyła i z niego sprowadza.

Dwie pierwsze z wymienionych wad usuwa przedstawione w następnym rozdziale rozwiązanie nazwane "DiscoLOGO prowizorium", złożone z nowego programu w Basicu współpracującego z Logo oraz z systemem AC-DOS, a ponadto z dwóch niestandardowych procedur Logo: DSAVE i DLOAD oraz z kopii interpretera Logo umieszczonej na dyskietce.

DiscoLOGO prowizorium

Instalując na dyskietce, zgodnie z podanymi dalej wskazówkami, Logo Sinclaira oraz program „Sprzęg DiscoLOGO AC-DOS" i procedury DSAVE i DLOAD algorytm komunikacji upraszczamy do postaci następującej:

I. Składowanie na dyskietce
1. Wydaj polecenie DSAVE podając w nim nazwę pliku i

nazwę (nazwy) procedur, które mają zostać w tym pliku przechowane (żądane procedury ukazują się na ekranie w polu edytora)

2. Zaakceptuj wyświetlone na ekranie procedury (CAPS-SHIFT + SYMBOL-SHIFT i C).

Po chwili, w czasie której DiscoLOGO m.in. zmniejsza obszar z procedurami do właściwych rozmiarów, pojawia się zgłoszenie Logo, co oznacza, że procedury są już zapisane na dyskietce. Można to sprawdzić, używając następującego ciągu instrukcji:

bye
GO TO 888
CAT

po których należy powrócić do Logo (instrukcją RUN).

II. Odzyskiwanie z dyskietki

1. Wykonaj polecenie DLOAD „nazwa-pliku

2. Po krótkiej chwili na ekranie pojawia się zgłoszenie Logo — wówczas wywołaj edytor instrukcją **edit** (bez parametru!)

3. Zaakceptuj sprowadzone z dyskietki procedury (CAPS-SHIFT + SYMBOL-SHIFT i C).

```
 1 REM Sprzeg discolo9o(->)dos
 2 REM ©ZPL IIUWr 851225
 6 LET swt=24839
 8 IF PEEK 23889=45 THEN  GO TO 500
10 IF PEEK swt=0 THEN  POKE swt,12
11 GO TO PEEK swt
12 GO TO 900: REM other "bye"
18 POKE swt,12: RANDOMIZE USR 24835: REM logout
20 LET l=50402: REM dsave
21 IF (PEEK l)=0 THEN  GO TO 23
22  LET l=l+1: GO TO 21
23 LET fna=49851: GO SUB 700
24 RANDOMIZE USR 15363: REM : SAVE f$CODE 50402,l-50402
26 GO TO 18
30 LET fna=49868: REM dload
31 GO SUB 700
34 RANDOMIZE USR 15363: REM : LOAD f$CODE
35 GO TO 18
500 CLS : POKE 23889,95
502  PRINT AT 3,3;"DiscoLOGO Prowizorium"
503 PRINT AT 5,3;" ©ZPL IIUWr 851224 **"
510 RANDOMIZE USR 15363: REM : LOAD "LOGO"CODE 24832
520  RANDOMIZE USR 24832
600 RANDOMIZE USR 15363: REM : LOAD "dsl"CODE 50402
650 RANDOMIZE USR 24835
700 POKE swt,0: LET f$="": FOR c=fna TO fna+9
701  LET f$=f$+CHR$ PEEK c: NEXT c: RETURN
888 RANDOMIZE USR 15360: REM  to dos!
889 RANDOMIZE USR 24832: REM lstart
900 REM not dsave nor dload
901 POKE swt,18
999  REM  ©ZPL
```

Program pośredniczący między Logo a systemem AC-DOS przedstawiono na wydruku 1. Pętla wyznaczająca adres, od którego można obciąć obszar roboczy edytora przed składowaniem (instrukcje: 20, 21, 22) stanowiła początkowo fragment procedury DSAVE. Okazało się jednak, że Basic wykonuje ją kilka razy szybciej niż Logo. Ponieważ zajmuje to sporo czasu w procesie składowania (zwłaszcza przy większej liczbie procedur), najlepiej procedurę tę wyrazić w kilku rozkazach kodu wewnętrznego.

```
TO DSAVE :a :b
MAKE "ln WORD :a "\ \ \ \ \ \ \ \ \ \
MAKE "lan 49851
REPEAT 10 [.deposit :lan first :ln
           make "lan :lan + 1
           make "ln butfirst :ln
          ]
RUN LIST "ed SE [] :b
.DEPOSIT 24839 20
BYE
END

TO DLOAD :b
MAKE "ln WORD :b "\ \ \ \ \ \ \ \ \ \
MAKE "lan 49868
REPEAT 10 [.deposit :lan first :ln
           make "lan :lan + 1
           make "ln butfirst :ln
          ]
.DEPOSIT 24839 30
BYE
END
```

Procedury DSAVE i DLOAD mają składnię wywołań identyczną jak procedury magnetofonowe SAVE i LOAD, a ich kod przedstawiono na wydruku 2. Użyto w nich dziwnych nazw, aby uniknąć kolizji z nazwami obiektów w innych procedurach. W razie potrzeby można nazwy te skomplikować jeszcze bardziej. Dostępne są tu wszelkie inne modyfikacje, bo DSAVE i DLOAD są zwykłymi, niestandardowymi procedurami Logo.

Instalowanie DiscoLOGO

Aby umożliwić posługiwanie się opisaną techniką wymiany informacji Logo z dyskietką, należy wykonać następujące czynności:

- wprowadzić do pamięci operacyjnej blok interpretera Logo z taśmy (drugi, najdłuższy z trzech) za pomocą polecenia
LOAD " " CODE

- przenieść go na dyskietkę za pomocą następujących poleceń systemu AC-DOS:

RANDOMIZE USR 15360 (zgłasza się DOS)
SAVE "LOGO" CODE 24832, 25000

- zapamiętać na dyskietce program organizujący sprzęg między Logo a systemem DOS, np. za pomocą polecenia AC-DOS:
SAVE „dlogo"

- wprowadzić Logo z dyskietki do pamięci operacyjnej pod nadzorem systemu DOS, tj. wydać polecenie:
RUN „dlogo"

- napisać procedury DSAVE i DLOAD

- umieścić na dyskietce procedury DSAVE i DLOAD za pomocą procedury DSAVE:

DSAVE "dsl [DSAVE DLOAD]

Uwaga. Przy dalszym korzystaniu, po sprowadzeniu Logo z dyskietki, w pamięci nie ma jeszcze procedur DSAVE i DLOAD. Sprowadzamy je bezpośrednio po zgłoszeniu się Logo lub w dowolnym późniejszym momencie pisząc:

bye
GOTO 600
edit

oraz akceptując uzyskaną zawartość obszaru edytora (CAPS-SHIFT + SYMBOL-SHIFT i C).

* * *

Logo w opisanym systemie ładuje się z dyskietki w ciągu około 8 sekund, a procedury — w zależności od rozmiaru i liczby — w ciągu kilku sekund. Składowanie trwa dłużej — do kilkunastu sekund, przy czym znaczną część czasu pochłania obcinanie obszaru edytora. Istotna wydaje się być również uwaga, że w opisanym rozwiązaniu — z chwilą pierwszego odwołania do procedury DSAVE i DLOAD — przestają działać standardowe procedury magnetofonowe. Tak więc, ewentualny kontakt z kasetami powinien poprzedzić użycie tych procedur (najlepiej cały własny „logokasetowy" dorobek przenieść na dyskietki).

Współpraca z dyskietkami jest oczywiście bardziej wydajna niż manipulowanie kasetami. Stwarza ona również nowe możliwości, np. szybkie łańcuchowe wykonywanie programów (ang. chaining). Należałoby życzyć sobie spopularyzowania tego nośnika przy naszych Spectrach. Mógłby to być także AC-DOS Apiny, byle niezawodny i tańszy!

Mówiąc o Logo Sinclaira warto dodać, że jeszcze w tym półroczu w wydawnictwach Uniwersytetu Wrocławskiego zostanie oddana do druku książka, w której pisze się o tym języku także krytycznie.

Nadesłano: 1986.01.26

EGZEMPLARZE ARCHIWALNE CZASOPISMA można nabywać za gotówkę w Klubie Prasy Technicznej w Warszawie, ul. Mazowiecka 12, tel. 27-43-65 oraz w Dziale Handlowym Wydawnictwa, ul. Bartycka 20, skr. poczt. 1004, 00-950 Warszawa, na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.

ZBIGNIEW SZKARADNIK
Bytom

Grafika żółwia w języku Forth

Grafika komputerowa jest tematem coraz bardziej aktualnym. Powoli zaczyna pojawiać się sprzęt posiadający pewne możliwości graficzne, lecz same możliwości sprzętowe to jeszcze nie wszystko. Niezbędne jest odpowiednio efektywne oprogramowanie. Pokutuje u nas od lat pogląd mówiący, że w celu wykorzystania możliwości graficznych sprzętu należy (i wystarczy!) stworzyć odpowiednią bibliotekę procedur graficznych, które mogą być następnie wykorzystane przez programy napisane w Fortranie lub Pascalu.

Uważam, że jest to błędne rozumowanie, gdyż klasyczny cykl produkcji oprogramowania (tzn. edycja — kompilacja — konsolidacja — uruchamianie) jest wybitnie nieefektywny w wypadku intensywnego wykorzystania grafiki. Należy sięgnąć po inne, bardziej wydajne narzędzia programowania, umożliwiające interakcyjne uruchamianie programów wykorzystujących grafikę. Chodzi tu przede wszystkim o interpretery języków programowania z wbudowaną grafiką. Takim narzędziem może być język Logo wyposażony w zestaw procedury realizujących tzw. grafikę żółwia (ang. turtle graphic) [4]. Jednakże grafika żółwia nie musi być zaletą wyłącznie języka Logo. Można ją wbudować także w interpreter innego języka programowania, o ile tylko gwarantuje on możliwość definiowania procedur rekurencyjnych. Może więc być to na przykład Lisp, Forth czy microProlog. W niniejszym artykule wyjaśniono, jak dobudować grafikę do języka Forth.

Wybór tego właśnie języka jest nieprzypadkowy i podyktowany został przede wszystkim dość znacznym jego rozpowszechnieniem, znacznie większą szybkością realizacji programów napisanych w języku Forth niż w innych językach (Lisp, Prolog) oraz możliwością prostej i naturalnej obsługi dołączanych (nawet nietypowych) urządzeń graficznych.

FILOZOFIA GRAFIKI ŻÓŁWIA

Grafika żółwia jest grafiką dwuwymiarową. Wykorzystuje układ współrzędnych biegunowych i jest oparta na umiejętnym użyciu kilku podstawowych operacji. Realizują je słowa (w terminologii języka Forth) umożliwiające [1]) kreślenie odcinków określonej długości do przodu (forward) i do tyłu (backward). Uzupełniają je operacje zmiany kierunku w lewo (left) i w prawo (right). Rysunek może być kreślony z podniesionym lub opuszczonym piórem (pen_up albo pen_down). Powyższy zestaw operacji umożliwia niezwykle naturalne konstruowanie obiektów graficznych. Trójkąt równoboczny można narysować na przykład przez trzykrotne powtórzenie sekwencji: narysowanie odcinka — zmiana kąta o 120 stopni:

```
: triangle (n — — —)
  3 0 do
     dup forward 120 left
     loop drop ;
200 triangle (kreśli trójkąt o boku 200)
```

W tabeli przedstawiono zestaw słów realizujących grafikę (słownik). Umożliwiają one podnoszenie i opuszczenie głowicy piszącej, przemieszczanie jej w określone położenie, kreślenie odcinków oraz zmianę kierunku ruchu.

IMPLEMENTACJA PAKIETU

W charakterze urządzenia graficznego sterowanego przez opisany pakiet programowy wykorzystano analogowy ploter typu 3078 japońskiej firmy Yokogawa. Został on dołączony do mikrokomputera Mera-60 w pracy dyplomowej [3]. Umożliwia sporządzanie rysunków w formacie 24 cm * 24 cm traktowanych przez komputer jako matryca 2400 * 2400 punktów.

Komputer „widzi" ploter jako zestaw czterech rejestrów o nazwach: X_register, Y_register, pen_counter oraz pen_status. Zawartość rejestrów X_register i Y_register sterują ruchem pióra przez przetworniki c/a. Zapis do rejestru Y_register powoduje podanie nowych współrzędnych XY do plotera. Rejestr pen_counter wykorzystywany jest do odmierzania czasu potrzebnego na przesunięcie pióra w określone położenie (przy użyciu słowa plot). Rejestr pen_status pełni różnorodne funkcje. Za jego pośrednictwem można opuszczać i podnosić pióro. Zawartość rejestru pen_status może być również testowana w celu potwierdzenia wykonania zadanej czynności przy użyciu słowa **wait**, umożliwiającego testowanie wybranych bitów. Przykładowo:

(1 wait) oznacza oczekiwanie na narysowanie odcinka elementarnej długości,

(4 wait) oczekiwanie na przesunięcie piórka w zadane położenie,
zaś

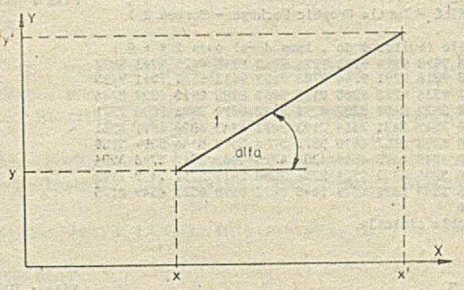
(8 wait) oczekiwanie na opuszczenie lub podniesienie pióra.

Aby w pełni wykorzystać zalety języka Forth do otrzymania szybkich procedur graficznych (grafika w języku Logo jest bardzo wolna, uniemożliwiająca praktycznie animacje rysunku), zdecydowano się na użycie jedynie liczb stałoprzecinkowych. Daje to spore korzyści przy sterowaniu monitora graficznego, choć oczywiście nie ma żadnego znaczenia w wypadku sterowania tak wolnego urządzenia jak ploter.

Obliczenia nowych współrzędnych XY dla układu współrzędnych biegunowych dokonuje się przy wykorzystaniu wzorów

$$x' = x + l * \cos(alfa)$$
$$y' = y + l * \sin(alfa)$$

gdzie **x**, **y** są współrzędnymi początku odcinka, x', y' — współrzędnymi końca odcinka, **l** jest długością odcinka, a **alfa** — kątem, pod którym jest rysowany (rys. 1).



Rys. 1. Obliczanie nowych współrzędnych XY

Wymaga to użycia funkcji trygonometrycznych sinus i cosinus (których nie ma w języku Forth). Należy więc zaimplementować je w sposób gwarantujący możliwie maksymalną szybkość. Uzyskano to dzięki stablicowaniu funkcji sinus dla kątów z przedziału od 0 do 90 stopni.

Słowo **draw** umożliwia kreślenie elementarnej długości odcinków łączących sąsiednie punkty rastra w jednym z ośmiu podstawowych kierunków (rys. 2). Wykorzystuje się

[1]) Słowa i sekwencje słów Fortha w tekście ujęto w nawiasy.

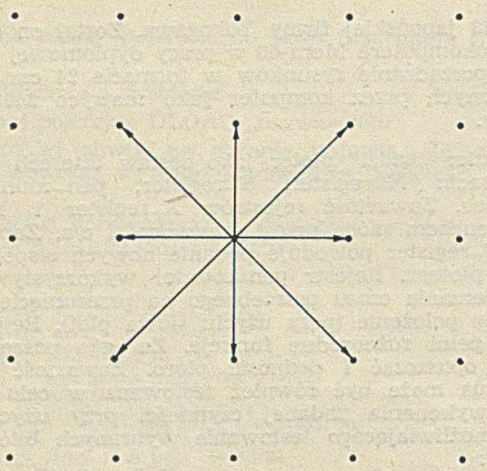
ją do kreślenia odcinków przy użyciu słowa **line.** Słowo **line** służy do kreślenia odcinka wyznaczając szereg kolejnych jego punktów, które z kolei są łączone za pomocą słowa **draw.** Obliczenia współrzędnych kolejnych punktów kreślonego odcinka dokonuje się według wzoru:

$$x'' = x + (y'' - y) * \text{ctg (alfa)}$$

lub

$$y'' = y + (x'' - x) * \text{tg (alfa)}$$

w zależności od tego, który przyrost współrzędnej jest większy (y czy x).



Rys. 2. Ilustracja kreślenia odcinków elementarnych (odległość między punktami rastra wynosi 0,1 mm)

W celu uproszczenia zapisu wyrażeń arytmetycznych, wprowadzono deklarujące zmienną słowo **var.** Dzięki niemu, na przykład, wyrażenie (AvBv + C!) można zapisać jako (AB + — >C). Słowo **recursive** umożliwia kompilację słów zdefiniowanych rekurencyjnie. Na wydruku przedstawiono pełny tekst źródłowy programu.

Wydruk. Pakiet grafiki w języku Forth

```
Screen # 1                                          File : TURTLE.FOH
 0 ( TURTLE - Turtle Graphic Package - Screen 1 )
 1
 2 0    variable  Xvar          : -)   1 Xvar ! ;
 3 : var (builds , does) Xvar @ if ! 0 Xvar ! else @ endif ;
 4
 5 octal
 6
 7 160520  constant  X_register   160522  constant  Y_register
 8 160524  constant  pen_counter  160526  constant  pen_status
 9
10 decimal
11
12   2400  constant  X_max         2400  constant  Y_max
13      0  var  X                    0  var  Y
14      0  var  direction            0  var  pen
15 --)

Screen # 2                                          File : TURTLE.FOH
 0 ( TURTLE - Turtle Graphic Package - Screen 2 )
 1
 2 : table (builds 0 do , loop does) swap 2 * + @ ;
 3 10000 9998 9994 9986 9976 9962 9945 9925 9903 9877
 4  9848 9816 9781 9744 9703 9659 9613 9563 9511 9455
 5  9397 9336 9272 9205 9135 9063 8988 8910 8829 8746
 6  8660 8572 8480 8387 8290 8192 8090 7986 7880 7771
 7  7660 7547 7431 7314 7193 7071 6947 6820 6691 6561
 8  6428 6293 6157 6018 5878 5736 5592 5446 5299 5150
 9  5000 4848 4695 4540 4384 4226 4067 3907 3746 3584
10  3420 3256 3090 2924 2756 2588 2419 2250 2079 1908
11  1736 1564 1392 1219 1045 0872 0698 0523 0349 0175
12  0000
13 91 table sintable
14
15 --)

Screen # 3                                          File : TURTLE.FOH
 0 ( TURTLE - Turtle Graphic Package - Screen 3 )
 1
 2 : \ ( --- ) in @ c/l / 1+ c/l * in ! ; immediate
 3
 4 : s180 dup 90 ) if 180 swap - endif sintable ;
 5
 6 : sin       \    ( alfa --- 10000 * sin (alfa) )
 7   360 mod dup 0( if 360 + endif
 8   dup 180 ) if 180 - s180 minus else s180 endif ;
 9
10 : cos       \    ( alfa --- 10000 * cos (alfa) )
11   90 + sin ;
12
13
14
15 --)

Screen # 4                                          File : TURTLE.FOH
 0 ( TURTLE - Turtle Graphic Package - Screen 4 )
 1
 2 : wait          ( n --- )
 3   begin dup pen_status @ and 0= until drop ;
 4
 5 : pen_up        ( --- )
 6   0 pen_status ! 0 -) pen 8 wait ;
 7
 8 : pen_down      ( --- )
 9   1 pen_status ! 1 -) pen 8 wait ;
10
11 : coordinates   ( x y --- )
12   -) Y -) X ;
13
14
15 --)

Screen # 5                                          File : TURTLE.FOH
 0 ( TURTLE - Turtle Graphic Package - Screen 5 )
 1
 2 : plot          ( x' y' --- )
 3   pen if pen_up 1 -) pen endif
 4   over over
 5   Y - abs swap X - abs max pen_counter !
 6   over over swap X_register ! Y_register !
 7   coordinates 4 wait ;
 8
 9 : draw          ( x' y' --- )
10   pen if pen_down endif
11   swap X_register ! Y_register ! 1 wait ;
12
13
14
15 --)

Screen # 6                                          File : TURTLE.FOH
 0 ( TURTLE - Turtle Graphic Package - Screen 6 )
 1
 2
 3
 4 : X'            ( l alfa --- x' )
 5                 \   x' = x + l * cos (alfa)
 6   cos m* 10000 m/ swap drop X + ;
 7
 8
 9 : Y'            ( l alfa --- y' )
10                 \   y' = y + l * sin (alfa)
11   sin m* 10000 m/ swap drop Y + ;
12
13
14
15 --)

Screen # 7                                          File : TURTLE.FOH
 0 ( TURTLE - Turtle Graphic Package - Screen 7 )
 1
 2
 3 0  var  delta_X   0  var  delta_Y
 4
 5
 6 : X"            ( y" --- x" )
 7                 \ x" = x + (y"-y) * ctg (alfa)
 8   Y - delta_X m* delta_Y m/ swap drop X + ;
 9
10
11 : Y"            ( x" --- y" )
12                 \ y" = y + (x"-x) * tg (alfa)
13   X - delta_Y m* delta_X m/ swap drop Y + ;
14
15 --)

Screen # 8                                          File : TURTLE.FOH
 0 ( TURTLE - Turtle Graphic Package - Screen 8 )
 1
 2 : line          ( x' y' --- )
 3   over over over over Y - abs swap X - abs swap
 4   ) if drop dup X ) if
 5                    1+ X do i i Y" draw 1 +loop
 6                  else
 7                    1- X do i i Y" draw -1 +loop
 8                  endif
 9     else swap drop dup Y ) if
10                    1+ Y do i X" i draw 1 +loop
11                  else
12                    1- Y do i X" i draw -1 +loop
13                  endif
14     endif coordinates  ;
15 --)

Screen # 9                                          File : TURTLE.FOH
 0 ( TURTLE - Turtle Graphic Package - Screen 9 )
 1
 2 : forward       ( l --- )
 3   dup 0= if
 4            drop
 5          else
 6            dup  direction X'
 7            swap direction Y'
 8            over over
 9            Y - swap X - -) delta_X -) delta_Y
10            line
11          endif ;
12
13 : backward      ( l --- )
14   minus forward ;
15 --)
```

```
Screen # 10                                          File : TURTLE.FOH
 0 ( TURTLE - Turtle Graphic Package - Screen 10 )
 1
 2 : left            ( alfa --- )
 3   direction + 360 mod -) direction ;
 4
 5 : right           ( alfa --- )
 6   minus left ;
 7
 8 : ?break ?terminal if sp! quit endif ;
 9
10 : home X_max 2/ Y_max 2/ plot 90 -) direction 1 -) pen ;
11
12 : recursive swudge [compile] [compile] swudge ; immediate
13
14
15 --)

Screen # 11                                          File : TURTLE.FOH
 0 ( TURTLE - Turtle Graphic Package - Screen 12 )
 1
 2 : ellipse         ( a b --- )
 3   over over
 4   swap 90 */
 5   swap 90 */
 6   Y )r X )r
 7   plot
 8   r) r) coordinates
 9   453 90 do
10         over i */
11         over i */
12         draw
13     loop drop drop
14   X Y  plot ;
15 --)

Screen # 12                                          File : TURTLE.FOH
( TURTLE - Turtle Graphic Package - Screen 12 )
 0
 1 : circle          ( r --- )
 2   dup ellipse ;
 3
 4 : arc             ( l alfa --- )
 5   swap over abs / swap
 6   dup 0< if
 7            abs 0 do dup forward -1 turn loop
 8          else
 9            0 do dup forward 1 turn loop
10          endif drop ;
11
12 : vector          ( l alfa --- )
13   -) direction forward ;
14
15
```

Przedstawiony pakiet realizujący grafikę żółwia ma pewne wady. Nieokreślone jest działanie programu w wypadku przekroczenia efektywnego zakresu rysowania (błędu). Zastosowanie liczb stałoprzecinkowych prowadzi również do powstania błędów wynikłych z obcinania ułamkowej części liczby (wskutek tego odcinki o długości poniżej 1 jednostki rastra, czyli 0,1 mm nie są w ogóle rysowane).

Wad tych pozbawiony jest opracowany na podstawie tego pakietu język graficzny, cechujący się ponadto wieloma udoskonaleniami i rozszerzeniami. Obejmują one:

- zestaw do transformacji rysunku (obrót, przesunięcie, powiększenie, pomniejszenie, wydłużenie i rozszerzenie),
- możliwość wyprowadzania i transformacji tekstów alfanumerycznych,
- możliwość definiowania własnych generatorów znaków alfanumerycznych,
- alternatywne tryby obsługi błędów (okno, zawijanie etc.),
- możliwość definiowania własnej obsługi błędów,
- możliwość sterowania wieloma urządzeniami graficznymi (ploter, monitor graficzny, drukarka),
- możliwość dołączania i obsługi własnych (nawet nietypowych) urządzeń graficznych bez rekompilacji systemu (!),
- zestaw plików zawierających definicje symboli używanych w wybranych dziedzinach (np. elektronice),
- możliwość określania szybkości rysowania i dokładności rysunku (dla plotera),
- możliwość pracy z dużą szybkością przy zastosowaniu liczb stałoprzecinkowych (przy poprawionej dokładności) lub z dużą dokładnością przy zastosowaniu liczb zmiennoprzecinkowych.

Wykaz słów realizujących grafikę (w nawiasach podano operacje wykonywane na stosie)

PEN_UP (———)
Podniesienie pióra

PEN_DOWN (———)
Opuszczenie pióra

PLOT (X Y ———)
Przesunięcie (uniesionego) pióra do punktu o współrzędnych XY

DRAW (X Y ———)
Rysowanie elementarnego odcinka łączącego punkt, w którym aktualnie znajduje się pióro, z punktem o współrzędnych XY (w wypadku plotera ruch będzie odbywał się wzdłuż jednego z podstawowych ośmiu kierunków, co w efekcie może spowodować narysowanie łamanej)

LINE (X Y ———)
Rysowanie odcinka łączącego punkt, w którym aktualnie znajduje się pióro, z punktem o współrzędnych XY (umożliwia wykreślenie odcinków w dowolnym kierunku)

FORWARD (L ———)
Rysowanie odcinka o długości L do przodu

BACKWARD (L ———)
Rysowanie odcinka o długości L do tyłu

LEFT (ALFA ———)
Obrót o kąt ALFA w lewo

RIGHT (ALFA ———)
Obrót o kąt ALFA w prawo

HOME (———)
Przygotowanie do rysowania (pióro ustawia się na środku rysunku z jednoczesnym przyjęciem kierunku rysowania w górę)

?BREAK (———)
Umożliwia przerwanie rysowania przez naciśnięcie dowolnego klawisza terminala

RECURSIVE (———)
Umożliwia kompilację słów zdefiniowanych rekurencyjnie (używane w sekwencji : XXX ... RECURSIVE XXX ... ;)

VECTOR (L ALFA ———)
Rysowanie wektora o długości L (pod kątem ALFA), rozpoczynającego się w punkcie, w którym aktualnie znajduje się pióro

ARC (L ALFA ———)
Rysowanie łuku o długości L i kącie ALFA, rozpoczynającego się w punkcie, w którym aktualnie znajduje się pióro

ELLIPSE (A B ———)
Rysowanie elipsy o osiach 2A i 2B oraz środku w punkcie, w którym aktualnie znajduje się pióro

CIRCLE (R ———)
Rysowanie okręgu o promieniu R oraz środku w punkcie, w którym aktualnie znajduje się pióro.

LITERATURA

[1] Biesiada H., Petrykowski W., Szkaradnik Z.: Opis języka Forth-60. Gliwice, 1984

[2] Brodie L.: Starting FORTH. Prentice Hall, Englewood Cliffs (NJ), 1981

[3] Struska M.: Podłączenie pisaka XY do maszyny Mera-60. Praca dyplomowa, Instytut Informatyki Politechniki Śląskiej, Gliwice, 1985

[4] Waligórski S.: Język LOGO. Informatyka, nr 7—9, 1985.

Nadesłano: 1986.02.17

KRZYSZTOF PERYCZ
Instytut Łączności
Oddział Gdańsk

Mikrokomputerowy system operacyjny SI (2)

W pierwszej części artykułu przedstawiono opis ogólnych właściwości systemu oraz jego poleceń. Poniżej omówiono zmienne systemowe, organizację systemu plików, algorytmy szeregowania i mechanizmy komunikacji zadań oraz zasady kontroli dostępu użytkowników.

ZMIENNE SYSTEMOWE

Zmienne systemowe (ang. system variables) zawierają wartości (parametry) związane z informacjami systemowymi. Umożliwiają one modyfikowanie zachowania systemu operacyjnego bez konieczności generowania go od nowa. Polecenie **set** działa na te parametry, które są możliwe do ponownego zdefiniowania przez użytkownika. Polecenie **display** powoduje wyprowadzenie ich aktualnych wartości. Dostęp do aktualnych wartości parametrów systemu mają także programy.

Poniżej podano nazwy i funkcje odpowiednich zmiennych systemowych:

abbreviation — pozwala na definiowanie skrótów dla nazw poleceń,

abbreviations — umożliwia korzystanie z uprzednio zdefiniowanych skrótów nazw poleceń,

command processor — definiuje stosowany procesor poleceń; może być on wymieniony na dowolny inny program,

date — podaje aktualną datę

disk space — podaje ilość wolnego miejsca na dowolnym zamontowanym woluminie,

implicit command list execution — pozwala na wykonanie listy poleceń tak, jakby stanowiła pojedyncze polecenie,

message queue information — podaje atrybuty wszystkich kolejek komunikatów; w systemach wielodostępnych wymagany jest specjalny przywilej, by przeglądać kolejki komunikatów innych użytkowników,

prompt — zawiera komunikat zaproszenia do wprowadzania danych z terminala; normalnie pojawia się on w postaci kropki, może być jednak rozszerzony, aby zawierał nazwę bieżącego skorowidza; komunikat zaproszenia może być wyświetlany na ekranie z inną intensywnością w celu odróżnienia od reszty tekstu,

ready — zawiera komunikat o gotowości, wyprowadzany po każdorazowym zrealizowaniu polecenia; zwykle jest to komunikat pusty, może jednak zawierać dane o zużyciu zasobów systemu, takich jak czas procesora czy pamięć operacyjna; podobnie jak komunikat zaproszenia, może odróżniać się od pozostałego tekstu intensywnością wyświetlania,

synonym — pozwala na zdefiniowanie synonimów nazw poleceń,

synonyms — umożliwia korzystanie z uprzednio zdefiniowanych synonimów nazw poleceń,

tasks — informuje o stanie aktywnych zadań,

time — podaje aktualny czas,

users — informuje o liczbie i nazwach użytkowników aktualnie zgłoszonych do systemu.

WOLUMINY I PLIKI

Wolumin, na którym mogą być przechowywane dane, posiada nazwę liczącą od 1 do 64 znaków oraz dwa atrybuty dostępności:

- **public**, gdy wolumin jest ogólnie dostępny,
- **private**, gdy jest dostępny jedynie dla tych użytkowników, którzy go imiennie zażądali.

Oprócz tego, wolumin ma także wszelkie atrybuty pliku.

W wypadku woluminów wymiennych, fakt ich zmiany jest rozpoznawany automatycznie. Gdy nastąpiła zmiana woluminu, który był w trakcie użytkowania, możliwe jest wybranie jednej z możliwych dróg wyjścia z sytuacji awaryjnej.

Plik stanowi nazwany obszar na woluminie. Nazwa pliku może mieć długość od 1 do 64 znaków. W skład nazwy mogą wchodzić dowolne znaki, z wyjątkiem znaku "/" — separatora ścieżki dostępu do pliku. Najprostsza ścieżka dostępu składa się z pojedynczej nazwy pliku. Bardziej skomplikowane ścieżki określają urządzenie, skorowidz, skorowidz podrzędny itd., aż do nazwy żądanego pliku.

W systemie S1 nie zakłada się istnienia jakiejkolwiek szczególnej struktury pliku. Taki plik może zawierać do 4 mld bajtów. Jeśli plik ma być traktowany w sposób specjalny, np. jako program, skorowidz lub plik indeksowy, to musi posiadać stosowany format, narzucony jednak tylko przez te części systemu operacyjnego, które z tego pliku mają korzystać.

Przestrzeń na urządzeniu pamięci masowej może być przydzielona plikom w trojaki sposób:

— jako porcja przyległych bloków,
— jako ciąg powiązanych bloków (często nieprzyległych do siebie),
— jako ciąg rozłącznych porcji, składających się z różnej liczby przyległych bloków.

Pierwsza metoda przydziału jest używana rzadko, tylko w wypadkach, gdy w grę wchodzi konieczność zagwarantowania odpowiedniego czasu dostępu do pliku, np. w systemach pracujących w czasie rzeczywistym. Istnieje możliwość przydzielania przestrzeni w kolejnych porcjach. Zapełnienie porcji może być sygnalizowane programowi zapisującemu na plik. Rozszerzenie pliku o nowe porcje może być dokonywane zarówno przez program użytkowy, jak i przez system operacyjny.

Druga z wymienionych metod przydziału jest używana najpowszechniej. Każdy z bloków wchodzących w skład pliku jest powiązany ze swoim poprzednikiem i następnikiem oraz z zapisem w skorowidzu, opisującym plik. Jest to bardzo niezawodny mechanizm, gdyż na ogół umożliwia odtworzenie zawartości pliku, nawet w wypadku zniszczenia skorowidza.

Trzecia metoda jest poniekąd kombinacją dwóch poprzednich. Gdy wymagane jest rozszerzenie pliku, przydziela się mu jeden lub więcej przyległych bloków z wolnej przestrzeni i zaznacza ten fakt w mapce przestrzeni pliku. W celu zwiększenia niezawodności można utrzymywać również duplikat tej mapki.

Sposób korzystania z pliku określają jego atrybuty:

read — umożliwiający odczytanie pliku,

write — pozwalający na zapisanie pliku,

execute — określający, że plik zawiera wykonywalny program,

append — umożliwiający dopisywanie do pliku, ale nie zapisywanie go na nowo od początku.

W wersjach wielodostępnych dostęp do poszczególnych plików może być dozwolony tylko konkretnym użytkownikom lub grupom użytkowników. Początkowo właścicielem pliku jest też użytkownik, który go utworzył. Prawo własności może być przekazywane innym użytkownikom.

Pliki mogą być przetwarzane w dwóch podstawowych trybach:

— strumieniowym, w którym plik jest traktowany jak ciąg bajtów,
— rekordowym, w którym plik stanowi ciąg rekordów, z których każdy jest ciągiem bajtów.

Plikami specjalnego przeznaczenia są skorowidze i pliki indeksowe. Skorowidze zawierają informacje o położeniu plików, ich atrybutach i datach związanych z ich wykorzystaniem, dotyczących np.:

- utworzenia pliku,
- ostatniego dostępu do pliku,
- ostatnio dokonanej modyfikacji,
- ostatnio przeprowadzonej archiwacji pliku.

Istnieje szereg różnych formatów skorowidzów, począwszy od specyfikujących urządzenia z nazwanym woluminem aż do umożliwiających realizację hierarchicznej struktury drzewiastej.

Plik indeksowy zawiera klucze i odpowiadające im informacje wskaźnikowe. Jest on skojarzony ze swym plikiem podstawowym, którego dotyczą wspomniane informacje wskaźnikowe. Oba te pliki łącznie mogą być traktowane jako jeden plik o dostępie kluczowym. Informacja w takim pliku może być zapisywana lub odczytywana w miejscu, które wskazuje podany klucz. Plik indeksowy może obsługiwać szereg formatów kluczy. Z jednym plikiem podstawowym może być skojarzona większa liczba plików indeksowych. Odpowiada to metodzie dostępu zwanej ISAM (ang. indexed sequential ascess method). W odróżnieniu od większości tradycyjnie stosowanych rozwiązań dostępu do plików kluczowych, pliki indeksowe mogą być przetwarzane jako zwykłe pliki, w oderwaniu od swych plików podstawowych.

ALGORYTMY SZEREGOWANIA

W systemie S1 dostępne są różne algorytmy szeregowania zadań, określające, które z zadań ma być wybrane do wykonania, gdy procesor stanie się dostępny.

W systemie bez przerwań, procesor może zacząć obsługiwać inne zadanie dopiero wtedy, gdy aktualnie wykonywane zgłasza się z żądaniem obsługi zewnętrznej. W celu zapewnienia wszystkim zadaniom bardziej równomiernego dostępu do procesora stosuje się specjalne wywołania systemowe, tzw. żądania zwolnienia procesora (ang. relinquish).

W systemie z przerwaniami najprostszy algorytm szeregowania przydziela wszystkim zadaniom równe porcje czasu, a wszystkie aktywne zadania wykonywane są kolejno. Bardziej skomplikowany algorytm uwzględnia dodatkowo priorytet zadania. Gotowe zadanie o wyższym priorytecie zawsze przejmuje procesor od zadania o priorytecie niższym; w ramach tego samego poziomu priorytetu zadania wykonywane są kolejno. Najbardziej skomplikowany algorytm szeregowania dzieli zadanie na dwie klasy: **zadania interakcyjne** oraz **wsadowe**. Za **zadania wsadowe** uznaje się takie zadania, które realizując obliczenia nie przesyłają żadnych informacji na terminal. Priorytet zadania ustala algorytm szeregowania na podstawie analizy zachowania się zadania, w celu zapewnienia **zadaniom interakcyjnym** pożądanego czasu odpowiedzi.

Jak każdy z elementów składowych systemu S1, algorytm szeregowania może zostać zastąpiony prywatnym algorytmem opracowanym przez użytkownika.

KOMUNIKACJA I SYNCHRONIZACJA MIĘDZYZADANIOWA

W systemie S1 istnieją mechanizmy, umożliwiające równolegle wykonującym się zadaniom:

- wymianę informacji,
- zabezpieczenie się przed konfliktowym dostępem do zasobów systemu,
- synchronizację z innymi zadaniami,
- reakcję na określone zdarzenia asynchroniczne.

Wymiana informacji między zadaniami jest realizowana za pośrednictwem komunikatów. Składają się one z nazwy (do 64 dowolnych znaków) i treści o dowolnej długości. Nazwa komunikatu może służyć jako jedyne źródło przekazywanej informacji. Komunikat może zostać skierowany do określonej kolejki komunikatów, istniejącej w dowolnym miejscu sieci komputerowej. Stosownymi wywołaniami systemowymi komunikat można utworzyć, zniszczyć, wysłać, odebrać, przekazać dalej lub zmodyfikować. Mechanizm ten jest zasadniczo przewidziany do przesyłania danych między użytkownikami różnych komputerów w sieci, a także komunikacji ze spoolerami oraz — w poczcie elektronicznej. Komunikaty tej samej kategorii mogą być grupowane w kolejkę, identyczną przez nazwę (do 64 znaków). Zawartość kolejki może być zrealizowana, zmodyfikowana, przesłana w inne miejsce lub zniszczona odpowiednimi wywołaniami systemowymi.

Zabezpieczenie przed konfliktem dostępu do zasobów systemu realizują blokady (ang. locks). Identyfikatorem blokady jest adres w pamięci, udostępniany przez wywołanie systemowe **create lock** (utwórz) lub **find lock** (odszukaj). Z blokadą może być opcjonalnie skojarzony jej kwalifikator, stanowiący ciąg o długości do 64 znaków (używany m.in. przy wyszukiwaniu blokady lub badaniu jej stanu). Blokady umożliwiają rozróżnienie trzech stanów:

- zabraniającego dostępu „na wyłączność", lecz zezwalającego na wspólne użytkowanie zasobu,
- zabraniające wszelkiego dostępu,
- zezwalające na dowolne korzystanie z zasobu.

Zmiany stanów są realizowane za pośrednictwem wywołań systemowych; istnieje możliwość oczekiwania do momentu zwolnienia się zasobu lub — badania czy jest on dostępny.

Synchronizacja międzyzadaniowa jest realizowana za pomocą semaforów. Podobnie jak blokady, są one identyfikowane przez adres w pamięci. Semafor może być w jednym z trzech stanów:

— **początkowym**, który trwa do momentu zakończenia pewnej operacji,
— **końcowym**, po jej pomyślnym zakończeniu,
— **błędnym końcowym**, po jej nieudanym zakończeniu.

Wywołania systemowe umożliwiają utworzenie semafora, badanie lub zmianę jego stanu, i oczekiwanie na moment jego przejścia w któryś stan końcowy.

Obsługa zdarzeń asynchronicznych jest w systemie S1 możliwa za pośrednictwem definiowalnych warunków (ang. conditions). Warunek jest określony przez jego nazwę (do 64 znaków), kod z przedziału (1, 65535) oraz standardową akcję podejmowaną przez system w momencie wystąpienia zdarzenia. Głównym zastosowaniem warunków jest sygnalizacja błędów oraz zdarzeń wyjątkowych, jak np. natrafienie na koniec pliku. Wywołania systemowe pozwalają na tworzenie, modyfikowanie lub usuwanie warunków. Możliwe jest też definiowanie specjalnych procedur obsługi danego warunku, sygnalizowanie systemowi jego wystąpienia oraz odczytywanie kodu ostatnio zasygnalizowanego warunku.

KONTROLA UPRAWNIEŃ UŻYTKOWNIKÓW

Istniejące w systemie S1 pojęcie użytkownika umożliwia klasyfikację osób korzystających z systemu (indywidualnie lub grupowo), w celu kontroli ich uprawnień. Użytkownikowi lub grupie użytkowników mogą zostać nadane przywileje zezwalające na korzystanie z funkcji systemu, które nie są normalnie ogólnie dostępne. Stosowany podział wyróżnia następujące klasy użytkowników systemu:

manager — umożliwia definiowanie nowych użytkowników bądź usuwanie istniejących,

operator — pozwala na bezpośrednie zarządzanie urządzeniami wejścia-wyjścia oraz programami spoolingu,

system programmer — pozwala na sterowanie zadaniami innych użytkowników; w systemach z ochroną pamięci dodatkowo umożliwia zmianę zawartości jakiejkolwiek jej komórki,

programmer — umożliwia odróżnianie użytkowników, którzy mają prawo używania niektórych wyróżnionych poleceń, jak erase czy initialize, od tych, którzy tego prawa nie mają,

application user — uniemożliwia wykonanie różnych działań destrukcyjnych, mogących spowodować zakłócenia pracy innych użytkowników.

Każdy z użytkowników może znajdować się w stanie:
free — zezwalającym na dowolne korzystanie z systemu w ramach posiadanych uprawnień,

Tabela 1. Porównanie właściwości systemów operacyjnych (+ oznacza istnienie danej cechy, — jej brak, ± istnienie jej częściowe, a ∞ brak ograniczeń)

1	S1	CP/M	MP/M II	CP/NET	Concurrent CP/M	MS-DOS	UNIX	XENIX	OASIS	Pick
	2	3	4	5	6	7	8	9	10	11
STRUKTURA SYSTEMU										
Przenośność										
Kod w języku wysokiego poziomu	+	+	+	+	+	—	+	+	+	—
Niezależność od urządzeń	+	—	—	—	—	—	+	+	+	+
Realizacja w języku maszynowym	+	+	+	+	+	+	+	+	+	+
Modularność										
Moduły wymienne	+	—	+	—	—	±	±	±	—	—
Moduły usuwalne	+	—	+	—	—	—	±	±	—	—
Rozszerzalność	+	—	+	—	—	—	—	—	—	—
Przesuwalność										
Regulowanie położenia w pamięci operacyjnej	+	—	+	+	+	—	+	+	—	+
PRZENOŚNOŚĆ PROGRAMÓW UŻYTKOWYCH										
Wywołania systemowe na poziomie źródłowym	+	—	—	—	—	—	+	+	—	+
Realizacja w języku maszynowym	+	+	+	+	+	+	+	+	+	—
Standardowe uporządkowanie bajtów w pliku	+	+	+	+	+	+	—	—	+	—
SYSTEM PLIKÓW										
Długość nazwy pliku (liczba znaków)	64	11	11	11	11	11	14	14	16	32K
Maks. wielkość pliku	∞	256 K	32M	32M	32M	4G	4G	4G	350 M	∞
Przydział pamięci dla pliku										
Porcje przyległych bloków	+	+	+	+	+	—	—	—	—	—
Pojedyncze bloki powiązane	+	—	—	—	—	—	—	—	—	—
Ciąg rozłącznych porcji	+	—	—	—	—	+	+	+	—	—
Zmienna długość rekordu	+	—	+	+	+	+	+	+	+	+
Kontrola dostępu do rekordu	+	—	+	—	+	—	—	+	+	+
Zarządzanie buforem we-wy przez system lub użytkownika	+	—	—	—	—	—	—	—	—	—
Hasła dostępu do plików	+	—	+	—	—	—	—	—	+	+
Sterowanie dostępem do wspólnych plików	+	—	—	—	—	—	+	+	+	+
Hierarchiczna struktura	+	—	—	—	—	—	+	+	—	+
Maks. liczba plików na dysku	∞	64	8K	8K	8K	64	∞	∞	∞	∞
Pliki o strukturze rekordowej	+	+	+	+	+	+	—	—	+	+
Maks. długość pliku	4G	128	2K	2K	2K	64K	—	—	350 M	32K
Pliki o dostępie swobodnym	+	+	+	+	+	+	—	—	+	+
Pliki o strukturze strumieniowej	+	—	—	—	—	—	+	+	+	—
Z dostępem swobodnym	+	—	—	—	—	—	+	+	+	—
Pliki z dostępem kluczowym	+	—	—	—	—	—	—	—	+	+

1	2	3	4	5	6	7	8	9	10	11
ISAM	+	—	—	—	—	—	—	—	+	—
VSAM	+	—	—	—	—	—	—	—	—	—
Drzewo binarne	+	—	—	—	—	—	—	—	—	—
PROCESOR POLECEŃ										
Wymienny	+	—	—	—	—	+	+	+	—	—
Wybór z menu	+	—	—	—	—	—	—	—	—	—
Konwencjonalny	+	+	+	+	+	+	+	+	+	+
Listy poleceń	+	+	+	+	+	+	+	+	+	+
Zmienne w listach	+	—	—	—	—	—	+	+	+	+
Wykonywanie warunkowe	+	—	—	—	—	—	+	+	+	+
Obsługa błędów	+	—	—	—	—	—	+	+	+	+
We-wy z (do) pliku	+	—	—	—	—	—	—	—	—	+
POLECENIA										
Tworzenie kopii i odtwarzanie jej	+	—	±	±	±	+	+	+	+	+
Porównywanie plików	+	—	—	—	—	+	+	+	+	—
Funkcje utajniania	+	—	—	—	—	—	+	+	+	—
Formatowanie dysku	+	—	—	—	—	+	+	+	+	+
Poczta elektroniczna	+	—	—	—	—	—	+	+	+	—
Weryfikacja systemu plików	+	—	—	—	—	+	+	+	+	+
Sortowanie	+	—	—	—	—	—	+	+	+	+
FUNKCJE NADZORCZE										
Wielodostępność	+	—	+	—	—	—	+	+	+	+
Wielużytkowość	+	+	+	—	+	—	+	+	+	+
Wielozadaniowość	+	—	+	—	+	—	+	+	—	—
Wieloprocesorowość	+	—	—	—	—	—	—	—	—	—
Liczba jednocześnie obsługiwanych procesorów	256	1	1	1	1	1	1	1	1	1
Semafory	+	—	+	—	+	—	—	—	—	—
Blokady	+	—	+	—	+	—	—	—	—	—
DZIAŁANIE W SIECI										
Zapewnione	+	+	—	+	—	—	—	—	+	—
Dostępne przez jakikolwiek program	+	—	—	+	—	—	—	—	—	—
Uwzględnione w systemie plików	+	—	—	+	—	—	—	—	—	—
OKNA PROGRAMOWE										
Dostępne	+	—	—	—	—	—	—	—	—	—
Przydział okien zadaniom	+	—	—	—	—	—	—	—	—	—
Okna nie związane z zadaniami	+	—	—	—	—	—	—	—	—	—
OBSŁUGA TERMINALI										
Terminale pełnoekranowe	+	—	—	—	—	—	—	—	+	—
Dostępne przez jakikolwiek program	+	—	—	—	—	—	—	—	+	—
Ekrany z odwzorowaniem punktowym	+	—	—	—	—	—	—	—	—	—
OBSŁUGA DRUKAREK										
Spooling	+	+	+	+	+	—	+	+	+	+
Drukowanie z odwzorowaniem punktowym	+	—	—	—	+	—	—	—	—	—

locked — zabraniającym korzystania z systemu, lecz z zachowaniem stanu posiadania,
deleted — w którym po użytkowniku istnieje tylko jego nazwa.

Z danym użytkownikiem skojarzonych jest wiele dodatkowych informacji używanych przez system, jak:

- nazwa skorowidza, w którym zaczyna się pracę bezpośrednio po zarejestrowaniu się w systemie poleceniem **logon**,
- typ standardowo używanego procesora poleceń,
- ustalone limity zasobów systemu, używanych stale lub chwilowo,
- przynależność do określonej grupy lub grup.

Pojęcie grupy użytkowników ułatwia kontrolę nad wspólnym dostępem do zasobów systemu i ich przydzielaniem.

W celu zapewnienia bezpieczeństwa systemu S1 kontroluje prawo dostępu do danych i programów.

Dostęp do plików może być kontrolowany na poziomie użytkownika, grupy użytkowników lub węzła sieci komputerowej. Wszystkie operacje na plikach w systemach wielodostępnych lub będących częścią sieci korzystają z mechanizmów ochraniających przed jednoczesną jego modyfikacją przez niezależnych użytkowników. Gdy plik jest otwierany lub też w dowolnym momencie przed jego zamknięciem, może zostać zablokowany dla zapisu lub nawet wszelkiego dostępu przez innych.

W systemie S1 istnieją również zabezpieczenia przed jednoczesnym wykonywaniem tego samego programu przez różnych użytkowników. Możliwe jest wszakże dzielenie wspólnej pamięci przez dwa zadania, nawet należące do różnych użytkowników.

Przedstawiony opis nie stanowi oczywiście pełnego obrazu systemu S1, gdyż trzeba by obszerniej omówić działanie bardziej rozbudowanych poleceń, takich jak: **copy, edit, link** czy **sort.** Więcej wypadałoby powiedzieć też o adaptowalności systemu, użytej w nim filozofii obsługi urządzeń wejścia-wyjścia, atrybutach programów, zadaniach i ich stanach, sposobach organizacji pamięci operacyjnej, operacjach synchronizujących, działaniu systemu w sieci, dostępnych operacjach arytmetycznych i różnorodnych konwersjach danych itp.

Różnorodność funkcji systemu S1 oraz jego wszechstronność na tle innych najpopularniejszych systemów operacyjnych przedstawiono w tabeli [1]).

[1]) Z przedstawionego porównania nie można wyciągać zbyt daleko idących wniosków, gdyż ma ono wyraźnie komercyjny charakter. Jest natomiast dobrym materiałem ilustracyjnym (przyp. red.).

LITERATURA

[1] Little J.: A Module Approach to Microcomputer Operating Systems. Computer Design, pp. 217—224, July 1984
[2] Multi Solutions Inc.: S1 Document S1-001-003. Lawrenceville (NJ), 1984
[3] Multi Solutions Inc.: S1 Highlights. Lawrenceville (NJ), 1984.

DINES BJÖRNER
Dansk Datamatik Center
Lyngby
Dania

Rola i zakres formalnej definicji Ady (2)

W drugiej części artykułu przedstawiono bardzo krótko opis podejścia obecnie przyjętego w opracowaniu formalnej definicji Ady oraz sformułowano i podsumowano wymagania stawiane tej definicji.

METODA D-SMoLCS

W metodzie D-SMoLCS (ang. Denotational Structured Monitored Linear Concurrent Systems) semantykę formalną Ady przedstawia się w dwóch hierarchicznie następujących po sobie krokach, tworząc:

- model denotacyjny,
- algebry semantyczne.

W celu zmodelowania wszystkich aspektów współbieżności i indeterminizmu [1], model denotacyjny wyraża się przy użyciu szeregu operatorów, takich jak „równolegle z" (ang. in parallel), „przeciążony" (ang. overloaded), „wybór" (ang. choice), „będący poprzednikiem" (ang. followed by) itp. — odpowiednio oznaczonych ("|", "+", "␣", ",", ";" itd.). Ten model może być oparty na wykorzystaniu mechanizmu wyjścia (ang. exit mechanism) w stylu imperatywnym lub aplikacyjnym (możliwym w języku Meta-IV), lub — na użyciu mechnizmu kontynuacji (także dopuszczalnym w języku Meta-IV). W konsekwencji, powstały model jest czytelny dla człowieka.

W drugim kroku przedstawia się denotacje utworzonego modelu. Ponieważ z jednej strony istnieją denotacje, a z drugiej — operacje na nich wykonywane (jak "|", "+", "␣", ",", ";" itd.), przedstawia się znaczenie tych operacji. Tego przedstawienia dokonuje się metodą wstępującą w pięciu fazach.

W fazie pierwszej, określającej podstawowy system przejść (ang. Basic Transition System), specyfikuje się, jakie są elementarne akcje indywidualnych procesów. W fazie drugiej, zwanej fazą synchronizacji, do podstawowego systemu przejść dodaje się reguły (parametry) określające punkty synchronizacji między procesami, tzn. definiuje się akcje atomowe (ang. atomic actions). W fazie trzeciej, zwanej fazą zrównoleglenia, do systemu synchronicznego dodaje się reguły (parametry) umożliwiające tworzenie procesów równoległych. W fazie czwartej, nadzorowania (ang. monitoring), dodaje się reguły (parametry) nakładające pewne ograniczenia na zachowanie procesów systemu równoległego. W ostatniej fazie, semantyki obserwacyjnej, dla systemu nadzorowanego, można otrzymać interpretację określonej semantyki na każdym z kilku poziomów abstrakcji: wejścia-wyjścia, przeplatania (ang. interleaving), scalania (ang. fair merging), rzeczywistego zrównoleglenia itd. Wykonuje się to przez odpowiednią parametryzację specyfikacji algebraicznych, otrzymanych dla operatorów synchronizacji, zrównoleglenia i nadzorowania.

Metodę SMoLCS można wykorzystać do formułowania specyfikacji SOS (ang. Structured Operational Semantics). W celu otrzymania jak najbardziej pożądanych właściwości semantyki obserwacyjnej używa się metod algebraicznych, choć możliwe jest także zastosowanie podejścia funkcjonalnego.

Obecnie planuje się podział definicji formalnej Ady na 6 części, tj. dwie główne specyfikacje semantyki statycznej i dynamicznej, z których każda będzie składać się oddzielnie ze specyfikacji dziedziny syntaktycznej i semantycznej oraz z definicji (równań) funkcji semantycznych.

Dziedziny zostaną wyspecyfikowane przy użyciu teorii Scotta, w notacji opartej na języku Meta-IV. Funkcje semantyki statycznej zostaną oparte na czystym, aplikacyjnym podzbiorze języka Meta-IV (tzn. bez użycia mechanizmu wyjścia), z możliwością algebraicznego definiowania pewnych dziedzin semantyki statycznej i operacji na ich obiektach. Semantykę dynamiczną planuje się oprzeć na metodzie D-SMoLCS. Ze względów praktycznych zostanie ona podzielona na trzy części, obejmujące różne aspekty Ady: sekwencyjność, wielozadaniowość i wejście-wyjście.

WYMAGANIA NAŁOŻONE NA DEFINICJĘ

Definicja formalna Ady powinna spełniać szereg wymagań, określających jej przydatność do celów wymienionych w pierwszej części tego artykułu. Są to wymagania bardzo różnorodne, lecz każde z nich w jakimś stopniu wpływa na przyjęte założenia i zakres definicji. Poniżej omówiono te wymagania, które wydają się najbardziej istotne.

Legalność. Przez legalność definicji formalnej Ady rozumie się jej opracowanie na podstawie umowy ważnej pod względem prawnym. Dzięki temu użytkownik uzyskuje pewność, że dokument uważany za definicję formalną jest tym samym dokumentem, który został dostarczony przez autorów.

Niesprzeczność i zupełność. Definicja formalna Ady powinna być niesprzeczna i zupełna tak, jak rozumie się te pojęcia w logice matematycznej.

Zrozumiałość i zwartość. Uważa się, że definicja jest zrozumiała, jeżeli dowolna osoba podejmująca się czytania definicji nie ma z tym nadmiernych trudności. Definicja formalna jest zwarta, jeśli jest precyzyjna, lecz niezbyt długa. Obie cechy definicji są względne, lecz nie zależą od stylu, w jakim jest napisana.

Poprawność. Definicja formalna funkcjonalnych aspektów języka jest poprawna, jeżeli spełnia wymagania projektodawcy tego języka. Najczęściej wymagania te są przedstawione w sposób nieformalny, dlatego ścisła poprawność definicji jest nieosiągalna. Co najwyżej można zakładać, że poprawna będzie taka definicja, która jest niesprzeczna i zupełna. Dlatego wprowadza się sformułowanie: wiarygodna poprawność (ang. believed correctness). Możliwość uznania definicji formalnej za wiarygodnie poprawną wynika stąd, że odzwierciedla ona w pewien sposób oficjalny dokument nieformalny. Znaczy to, że nie ma oczywistych rozbieżności między definicją formalną a oryginalnymi intencjami, zawartymi w wymaganiach projektodawcy lub wyróżnionymi w nieformalnej specyfikacji (tzn. w podręczniku języka — ang. Language Reference Manual, LRM).

Przejrzystość (ang. accessibility and referenceability). Dla osób programujących w Adzie i opracowujących jej translatory, język składa się z pewnej liczby uzgodnionych i werbalnie określonych konstrukcji (pojęć). Odpowiedzi na pytania dotyczące właściwości każdej z tych konstrukcji powinny być możliwe do odnalezienia w krótkim opisie na określonej stronie definicji formalnej, tzn. powinny być łatwo dostępne (ang. accessible). Jeżeli ten warunek jest spełniony, to można jednoznacznie odwoływać się (ang. refer) do tych stron definicji.

Permisywność (ang. permissiveness). Definicja formalna jest permisywna, jeżeli — w idealnym przypadku — wyraża wszystkie dopuszczalne aspekty: kolejności obliczania, optymalizacji, równoległości, indeterminizmu itd.

Niezależność implementacyjna. Uważa się, że definicja formalna jest wolna od obciążeń implementacyjnych, jeżeli

nie faworyzuje nadmiernie określonego sposobu implementacji, nie zamierzonego przez twórców języka.

Automatyzowalność. Definicja formalna Ady jest automatyzowalna, jeżeli nadaje się do przetwarzania komputerowego. Celem takiego postępowania może być ułatwienie:
— pielęgnacji i aktualizacji definicji formalnej Ady,
— skorelowania jej z innymi automatyzowalnymi dokumentami dotyczącymi Ady, jak podręcznik języka, ANNA, DIANA itp.,
— przeprowadzenia eksperymentów naukowych.

Korelowalność. Definicja formalna Ady jest korelowalna, jeżeli może być systematycznie, wyczerpująco i jednoznacznie odwzorowana na podręcznik języka (ang. language reference manual, LRM).

Pielęgnowalność. Definicja formalna Ady jest pielęgnowalna, jeżeli można dla niej opracować zestaw narzędzi służących np. do przeglądania definicji, do wyszukiwania związków między definicją formalną a podręcznikiem języka itp. Dobrym środowiskiem do wspomagania opracowywania narzędzie pielęgnacyjnych wydaje się system MENTOR [4].

Na definicję formalną Ady nakłada się jeszcze inne wymagania oprócz wymienionych tak, aby stanowiła podstawę do dalszych opracowań.

Podstawa opracowania procesora języka. Definicja formalna Ady jest odpowiednia jako podstawa do wytworzenia interpretera, kompilatora i systemu wspomagania, jeżeli dotyczy ono wszystkich aspektów języka i można to zapewnić używając tej definicji. Definicja formalna Ady jest odpowiednia do dowodzenia poprawności procesorów Ady, jeżeli można podać a priori konstruktywny dowód poprawności procesora Ady (a nie — posteriori — dla danego procesora). Tak więc, definicję formalną Ady rozumie się jako punkt wyjścia do opracowania rzeczywistego procesora. Proces opracowania rozumie się przede wszystki jako transformowanie, uściślanie i wzbogacanie definicji formalnej Ady w poszczególnych etapach, od projektu abstrakcyjnego, przez projekt szczegółowy, aż do implementacji (kodu).

Podstawa atestacji. Definicja formalna Ady jest odpowiednia do atestacji, jeżeli umożliwia skonstruowanie zestawu testów napisanych w Adzie, w celu przetestowania dowolnego kompilatora. Programy testowe powinny być „przezroczyste", tzn. programista powinien być przekonany, że są to rzeczywiste programy. Ponadto powinny one pokrywać cały zakres możliwości programowych. Powinny także pokrywać cały zakres implementacji, tzn. poddawać testowaniu nie tylko strukturę języka, lecz także możliwe do przewidzenia struktury procesorów. Przez badanie zgodności definicji formalnej Ady i zestawu testów ACVC (ang. Ada Compiler Validation Capability) rozumie się wykonanie jednej z następujących czynności:

1. Użycie rozumowania nieformalnego w celu stwierdzenia, że programy testowe z zestawu ACVC są przetwarzane poprawnie przy użyciu definicji formalnej Ady, lub wykazanie, że programy z tego zestawu nie są poprawnymi programami testowymi (tzn. są sprzeczne z intencjami projektantów języka).

2. Użycie technik dowodzenia formalnego do ręcznego zweryfikowania warunku 1.

3. Automatyczne użycie technik dowodzenia formalnego w celu zweryfikowania warunku 1.

4. Utworzenie prototypu (p. poniżej) do sprawdzenia warunku 1.

5. Użycie formalnych metod opracowywania kompilatorów, w celu sprawdzenia warunku 1.

6. Bezpośrednie wykonanie kodu samej definicji formalnej Ady w celu sprawdzenia warunku 1. Ponadto, możliwe jest podstawienie od tak rozumianej zgodności i zweryfikowanie zgodności definicji formalnej Ady z podręcznikiem języka.

Podstawa systemu dowodzenia poprawności. Mówiąc, że definicja formalna Ady może służyć jako podstawa dowodzenia poprawności programów napisanych w Adzie, zakłada się iż te programy są w pewien sposób wyspecyfikowane, tzn. istnieją pewne asercje określające ich właściwości, a definicja umożliwia formalne wykazanie spełnienia tych asercji. Zakłada się także, że definicja jest wyrażona w postaci systemu dowodzenia poprawności (zbioru takich systemów) lub — że system dowodzenia (ich zbiór) można wyprowadzić z tej definicji.

Podstawa tworzenia prototypów. Utworzenie prototypu polega na opracowaniu prototypowego translatora Ady i wiąże się z przekształceniem, o ile to możliwe, definicji formalnej Ady bezpośrednio na wykonywalny kod wysokiego poziomu, np. SETL [5]. Szybkie utworzenie prototypu polega na sprawnym i tanim opracowaniu wstępnej wersji programu w celu poddania go sprawdzeniu przez użytkowników.

Podstawa opracowywania dokumentacji. Definicja formalna Ady jest odpowiednia do pisania podręczników użytkowania języka itp., jeżeli może służyć do łatwego i systematycznego opracowania ich tekstów oraz umożliwia łatwe odwoływanie się do niej z tych tekstów, np. w celu sprawdzenia ich niesprzeczności i zupełności. Dokumentami opracowywanymi przy użyciu definicji mogą być podręczniki dla początkujących, dojrzałych lub zaawansowanych programistów, a także dla implementatorów, osób zajmujących się atestacją itd.

Przedstawione wymagania oparto na wielu założeniach. W pierwszym z nich zakłada się zupełność i niesprzeczność Ady. Jeżeli w czasie opracowywania definicji formalnej Ady okaże się, że to założenie nie jest spełnione, to należy określić procedurę rozstrzygania zupełności i niesprzeczności. Jeżeli w trakcie formułowania obecnej definicji formalnej zostanie we właściwy sposób stwierdzona sprzeczność lub niezupełność języka, to definicja nie zostanie określona — tak więc rozbieżność będzie nie rozstrzygnięta. W drugim założeniu przyjmuje się, że istnieją metody opracowania definicji formalnej, która będzie spełniała wszystkie powyższe wymagania. Jeżeli w czasie jej opracowywania okaże się, że nie jest to możliwe, to należy określić procedurę poszukiwania rozwiązań kompromisowych, ze względu na styl definicji lub użycie definicji złożonej z alternatywnych. W obecnym projekcie definicji formalnej Ady taką procedurą jest publiczne opiniowanie wstępnej wersji.

Z powyższych wymagań i założeń wynika wiele innych, pochodnych właściwości definicji formalnej Ady, jak np.: brak niejednoznaczności, nienadmierne wyspecyfikowanie, modularność, wyprowadzalność poradnika implementacyjnego, możliwość tworzenia narzędzi wspomagających itd.

UWZGLĘDNIENIE WYMAGAŃ

Wymagania wymienione w poprzednim punkcie należy uznać za idealne. Poniżej omówiono stopień ich uwzględnienia w bieżącym projekcie.

Legalność. Wydaje się, że jeszcze nie nadszedł czas, aby społeczność informatyczna (dostawcy i nabywcy oprogramowania) była zainteresowana definicją formalną Ady jako umową w sensie prawnym. Jeżeli zostaną spełnione wszystkie pozostałe wymagania, to można oczekiwać, że wymaganie legalności będzie pożądane. Spełnienie tego wymagania jest więc celowe.

Niesprzeczność i zupełność. Spełnienie tego wymagania jest najbardziej pożądane. Pewną przeszkodą może być jednak oficjalna interpretacja takich pojęć, jak: błędny, niezdefiniowany, współdzielony przez pragmę, permisywny itp., różna od interpretacji użytkowników definicji, tj. projektantów Ady, organizacje normalizacyjne, ośrodki atestowania, inne osoby zajmujące się formalizacją itp.

Zrozumiałość i zwartość. Spełnienie tych wymagań jest również ogromnie pożądane. Jednym ze sposobów osiągnięcia celu jest skorelowanie definicji formalnej z podręcznikiem języka. Inny sposób polega na wprowadzeniu adnotacji określających w języku angielskim sens formuł matematycznych. Proces korelowania definicji z podręcznikiem i wprowadzania adnotacji ma też wpływ na kształt samych formuł.

Poprawność (wiarygodna poprawność). Jest to również pożądana właściwość, którą można osiągnąć metodami podobnymi do proponowanych powyżej.

Przejrzystość. Spełnienie tego wymagania jest również pożądane w najwyższym stopniu.

Permisywność. Jeśli można zidentyfikować wszystkie tego rodzaju właściwości języka, określające permisywność ze względu na implementację i indeterminizm, to definicja formalna Ady musi również być permisywna.

Niezależność implementacyjna. Definicja formalna Ady będzie najprawdopodobniej ukierunkowana na modele. Defi-

nicje denotacyjne, tzn. ukierunkowane na modele, nie muszą jednak prowadzić do preferowania określonych metod implementacji. W obecnej definicji formalnej Ady należy dążyć do zachowania niezależności implementacyjnej, co wynika również z dążenia do permisywności.

Automatyzowalność. Obecna definicja formalna Ady, wraz z podręcznikiem języka, opisem stopnia jej korelacji z podręcznikiem, niezależnymi od podręcznika adnotacjami i opisem różnych, choć niekoniecznie wszystkich, aspektów semantyki języka specyfikacyjnego, będzie zautomatyzowana. Znaczy to, że będzie utworzony zestaw narzędzi komputerowych, służących do wspomagania odpowiednich czynności.

Korelowalność. Opracowywanej definicji formalnej Ady stawia się zdecydowane wymaganie, aby była ściśle, jasno i w pełni skorelowana z podręcznikiem języka.

Pielęgnowalność. Podobnie wyraźnie wymaga się, aby obecna definicja formalna Ady była pielęgnowalna, skąd wynika konieczność poddania jej automatyzacji.

Podstawa do opracowania procesora języka. W obecnym projekcie definicji formalnej Ady, opartym na podejściu ukierunkowanym na modele, lecz prowadzącym do abstrakcyjnego zdefiniowania języka, dąży się do utworzenia definicji, która mogłaby stanowić punkt wyjścia do opracowania interpretera, kompilatora lub innego narzędzia programowego. Dla klasycznych definicji denotacyjnych i operacyjnych, jak np. formalny opis Ady dokonany w Dansk Datamatik Center [3], istnieją dobrze znane metody [2], prowadzące do systematycznego i ścisłego opracowania kompilatorów na podstawie definicji języka. W obecnym projekcie dąży się do utworzenia definicji formalnej Ady, która mogłaby być podstawą do opracowania procesora języka, lecz nie określa się, jak formalnie wyprowadzić taki procesor z definicji. Jednakże, można uzasadnić taką możliwość odwołując się do abstrakcji, ukierunkowanej na model planowanej definicji formalnej. Autor jest przekonany, że planowana definicja formalna Ady umożliwi łatwe użycie do tego celu technik formalnych, jak transformacje, wzbogacanie i uściślenie lub pobudzi ich opracowanie.

Podstawa atestacji. Przewiduje się podjęcie badań mających na celu stwierdzenie, czy definicja formalna Ady może służyć jako podstawa do opracowania zestawu testów ACVC oraz czy może być użyta do dowodzenia właściwości tych programów. Nie zamierza się natomiast w tym projekcie podejmowania prób mających na celu sprawdzenie zgodności większej części zestawu testów ACVC z definicją formalną Ady (i odwrotnie). Tak więc, w zakres obecnego projektu wchodzi jedynie badanie podstawowych aspektów atestacji oraz badanie stosowalności nieformalnych i formalnych metod dowodzenia zgodności z definicją zestawu testów ACVC.

Podstawa systemu dowodzenia poprawności. Przewiduje się podjęcie badań mających na celu stwierdzenie możliwości wyprowadzania systemów dowodzenia z definicji formalnej Ady. Nie zakłada się jednak, że obecnie planowana definicja formalna Ady będzie musiała zapewniać utworzenie takiego systemu dowodzenia.

Podstawa tworzenia prototypu. W tym projekcie nie zakłada się, że obecna definicja formalna Ady będzie stanowić podstawę do szybkiego utworzenia prototypu, na przykład, interpretera. Nie planuje się również wykazania, że definicja ma tego rodzaju możliwości. Jednakże, z uwagi na możliwości użycia definicji formalnej do opracowania procesora języka (p. powyżej), dla obecnego projektu interesujące jest ustalenie stopnia korelacji między definicją formalną Ady a interpreterem NYU/Ada Ed (ang. New York University educational Ada interpreter) napisanym w języku SETL. Choć w obecnym projekcie nie przewiduje się podjęcia odpowiednich badań, to w wypadku ich niezależnego rozpoczęcia bardzo pożądane jest podjęcie współpracy.

Podstawa opracowania dokumentacji. Jest to podstawowe wymaganie, narzucone przez samych autorów definicji formalnej Ady, aby była ona wyrażona (przedstawiona) w sposób umożliwiający łatwe wyprowadzenie z niej szeregu podręczników na różnym poziomie trudności, przeznaczonych dla początkujących, dojrzałych, doświadczonych i zaawansowanych programistów, implementatorów, naukowców zajmujących się formalizacją itd. Jednakże, w obecnym projekcie przewiduje się systematyczne wyprowadzenie z definicji tylko jednego takiego dokumentu.

* * *

W niniejszym artykule przedstawiono dość pobieżne — mające raczej wyliczeniowy niż głębszy, analityczny charakter — rozważania dotyczące różnych aspektów języka Ada, prowadzące do sklasyfikowania i skwantyfikowania wielu wymagań, które powinna spełniać obecnie opracowywana definicja formalna Ady. Celem opublikowania artykułu jest poddanie tych rozważań pod osąd międzynarodowej społeczności zajmującej się Adą. Autor oczekuje więc poważnych komentarzy i prosi o gruntowną krytykę, a ze swej strony deklaruje gotowość do obiektywnej oceny nadesłanych uwag i ewentualnego włączenia ich do poprawionej, ostatecznej wersji tego raportu.

Tłum. i oprac.:
JANUSZ ZALEWSKI

LITERATURA

[1] Astesiano E. et al.: On the Parameterized Algebraic Specification of Concurrent Systems. Proc. TAPSOFT Conf., Berlin, March 1985, pp. 342—358. Lecture Notes in Computer Science, Vol. 185. Springer-Verlag, Berlin, 1985

[2] Bjørner D., Jones C. B.: Formal Specification and Software Development. Prentice-Hall, Englewood Cliffs (NJ), 1982 (chapter 9)

[3] Bjørner D., Oest O. N.: Towards a Formal Description of Ada. Lecture Notes in Computer Science, Vol. 98. Springer-Verlag, Berlin, 1980

[4] Kahn G. et al.: Natural Semantics on the Computer. Internal Report, INRIA, Rocquencourt, 24 May 1985

[5] Kruchten F., Schonberg E.: The Ada/Ed system — a largescale experiment in software prototyping using SETL. Technology and Science of Informatics. Vol. 3, No. 3, pp. 175—13, 1984.

Ogólnopolski Konkurs im. Jerzego Trybulskiego na najlepsze wdrożone prace z dziedziny zastosowań informatyki

W ostatnich latach znacznie zmniejszyła się u nas liczba wdrożonych rozwiązań dotyczących zastosowań informatyki w gospodarce. I Walny Zjazd Polskiego Towarzystwa Informatycznego, uwzględniając dotychczasowy stan zastosowań informatyki oraz efektywność jej wykorzystania w gospodarce, postanowił zorganizować konkurs, którego celem jest stymulacja szybszego rozwoju zastosowań informatyki w różnych dziedzinach gospodarki. Chcąc uczcić pamięć jednego z założycieli Towarzystwa oraz docenić Jego wielki wkład w rozwój zastosowań informatyki Zjazd postanowił nazwać konkurs imieniem Jerzego Trybulskiego.

Bezpośrednimi **celami konkursu** są:

- rozpowszechnienie w gospodarce wyróżniających się rozwiązań informatycznych
- stworzenie bodźców do podejmowania prac zmierzających do zwiększenia efektywności zastosowań informatyki.

Warunki konkursu

- Prace konkursowe mogą dotyczyć rozwiązań informatycznych wdrożonych w jednostkach gospodarki uspołecznionej i nie uspołecznionej w kraju i za granicą.
- Uczestnikami konkursu mogą być obywatele PRL, którzy są autorami prac wykonanych na terenie kraju.
- Oceniane będą tylko rozwiązania bieżące, eksploatowane przez użytkownika co najmniej od roku.
- Kryterium oceny prac będą uzyskane wymierne efekty ekonomiczne.

Prace konkursowe należy zgłaszać na formularzu z krótka dokumentacja rozwiązania w terminie do 30 września 1986 roku pod adresem:

Polskie Towarzystwo Informatyczne, Oddział we Wrocławiu
Plac Grunwaldzki 9, 50-370 Wrocław

Formularze „Karty zgłoszeń" można otrzymać w terenowym Kole PTI lub w Oddziale wrocławskim.

Szczegółowe informacje dotyczące warunków, nagród, organizacji konkursu itp. są podane w Regulaminie opublikowanym w Biuletynie PTI nr 9 z 1985 r. Regulamin ten jest również dostępny w każdym terenowym Kole PTI.

Oddział PTI we Wrocławiu
Przewodniczący Komisji Konkursowej
dr JAN SZTAJER

PIOTR ZASKÓRSKI
EWA KASPRZAK
Wojskowy Instytut Informatyki
Warszawa

Technologia wytwarzania oprogramowania użytkowego z komputerowym wspomaganiem

W praktyce projektowania i wytwarzania oprogramowania obowiązywała powszechnie i nadal obowiązuje zasada indywidualnego tworzenia całości programu przez każdego wykonawcę. Rozwój metod organizacji zespołów wykonawczych doprowadził do spopularyzowania i rozpowszechnienia modelu pracy w tzw. zespole programisty wiodącego (ang. chief programmer team [1]). Metody i techniki wspomagające ten model pracy nie nadążały jednak za potrzebami takiego zespołu. Co prawda, języki wyższego poziomu ułatwiają komunikację między członkami zespołu, lecz nie dają podstaw do tworzenia narzędzi standaryzacji pracy umożliwiających weryfikację jej wyników. Zespół programisty wiodącego działał więc stosując tradycyjne narzędzia wymuszające na jego liderze dbałość o spójność technologiczną wytworzonych części oprogramowania.

Na etapie projektu technologicznego programista wiodący nie miał narzędzia wspierającego pracę. W wypadkach sporadycznych, dla bardziej złożonych systemów wytwarzano tradycyjnie bibliotekę makrodefinicji. Umożliwiała ona jednolitą symbolizację, np. struktur danych oraz zawierała te fragmenty opisu systemu, które nosiły cechy powtarzalności proceduralnej lub strukturalnej.

Ważnym etapem w rozwoju technologii wytwarzania oprogramowania było przyjęcie koncepcji standaryzacji typowych szkieletów programów i podjęcie próby stworzenia stosownej biblioteki. Na korzystającym z niej programiście wymusza spójność konstrukcyjną, gdyż porządkuje strukturę programów. Stanowi również bardzo dobre narzędzie dydaktyczne w nauczaniu języka, w którym opracowano teksty źródłowe szkieletów.

Złożoność algorytmiczna programów i wynikająca stąd złożoność przepływu sterowania stanowią jednak istotne ograniczenia stosowalności tej metody. Najbogatsza bowiem biblioteka szkieletów nie określi prototypów dla konkretnych, trudnych do przewidzenia konstrukcji programowych. Stąd też kolejnym etapem rozwoju technologii jest wykorzystywanie języków opisu problemu i ich analizatorów do produkcji — w sposób adekwatny do potrzeb projektowanego systemu — pożądanych konstrukcji programowych w postaci biblioteki makrodefinicji.

TECHNIKI RODZAJOWE OPARTE NA PAKIECIE PSL/PSA

Biblioteka makrodefinicji jest uzyskiwana w sposób automatyczny, przy użyciu technik rodzajowych. Techniki rodzajowe stanowią kompleks narzędzi przeznaczonych dla zespołu programisty wiodącego, opartych na właściwościach komputerowego wspomagania procesu projektowania systemów informatycznych PSL/PSA i są oryginalnym dorobkiem zespołu autorskiego, stanowiąc narzędzie rozszerzające obszar stosowalności metodologii PSL/PSA poza „czyste projektowanie" systemów informatycznych. Metoda PSL/PSA wraz z uzupełniającym kompleksem technik rodzajowych stanowi całościowo spójne i użyteczne narzędzie, wspomagające budowę systemu informatycznego od etapu analizy systemu, przez utworzenie zadania projektowego i projektowanie koncepcyjne, aż do etapu projektowania technologicznego i doskonalenia systemu informatycznego.

Metoda PSL/PSA zapewnia utrzymywanie jednolitego konstrukcyjnie opisu projektu — tworzonego na podstawie opisu w języku PSL — w postaci komputerowej bazy danych. Baza danych projektu odzwierciedla poprawne formalnie i logicznie opisy, co jest właściwością analizatora PSA tej metody.

Metoda PSL/PSA w wersji 2.1, a szczególnie jej zakres funkcjonalny, warunkuje opłacalność stopnia dekompozycji prowadzonego opisu projektu. Szczegółowość opisu projek-

[1]) Inny odpowiednik polski — zespół głównego programisty (przyp. red.).

Dr inż. PIOTR ZASKÓRSKI ukończył w 1970 r. studia na Wydziale Cybernetyki Wojskowej Akademii Technicznej. W 1976 r. obronił rozprawę doktorską z dziedziny modelowania matematycznego z wykorzystaniem techniki obliczeniowej. W całym okresie pracy zawodowej projektował i oprogramowywał użytkowe systemy informatyczne. Od 1980 r. zajmuje się komputerowym wspomaganiem projektowania i wytwarzania technologii systemów informatycznych opartym na pakiecie PSL/PSA i możliwościami jego rozbudowy. Obecnie pracuje nad wykorzystaniem sprzętu mini- i mikrokomputerowego w zarządzaniu.

Mgr EWA KASPRZAK ukończyła studia na Wydziale Matematyki, Mechaniki i Informatyki Uniwersytetu Warszawskiego w 1980 r. Od początku swojej pracy zawodowej zajmuje się wytwarzaniem oprogramowania użytkowego oraz implementacją narzędzi wspomagających ten proces. Ma w dorobku oryginalną implementację techniki PSL/PSA na komputer R-32 w wersji wsadowej i interakcyjnej. Obecnie zajmuje się zastosowaniem wytworzonych narzędzi w praktycznym działaniu.

tu jest stymulowana jego użytecznością. Pracując ze standardową wersją pakietu, celowe jest prowadzenie opisu projektu do poziomu założeń programowych bez wnikania w treść pojedynczego programu. Przy posługiwaniu się technikami rodzajowymi zasadna staje się dalsza dekompozycja projektowanego systemu, z uwagi na możliwość uzyskiwania gotowych produktów programistycznych.

Techniki rodzajowe, stanowiące rozszerzenie standardowej wersji pakietu PSL/PSA, są wzajemnie uzupełniającymi się pod względem funkcjonalnym raportami włączonymi pod wspólne sterowanie PSL/PSA i mają postać generatorów:

— opisu struktur danych w języku COBOL
— szkieletów programów w języku COBOL.

COBOL nie stanowi istotnego ograniczenia dla idei technik rodzajowych. Z uwagi na stopień rozpowszechnienia tego języka, techniki te są bezpośrednio użyteczne. Posługując się konstrukcjami takich języków jak Pascal lub PL/I można uzyskać bogatsze pod względem strukturalnym elementy oprogramowania użytkowego, generowanego w tych językach. Z chwilą zaistnienia odpowiednich potrzeb istnieje możliwość wytworzenia generatorów dla języków o bogatszych strukturach danych i rozbudowanych konstrukcjach sterowania.

W warunkach stosowania technik rodzajowych jako kompleksu narzędzi połączonych funkcjonalnie z pakietem PSL/PSA, programista wiodący projektowanego systemu powinien być członkiem zespołu projektowego, począwszy od fazy opracowywania koncepcji technologii przetwarzania. Cały opis projektu winien być prowadzony w standardzie projektowym, jaką jest metoda PSL/PSA. Definiowanie koncepcji przetwarzania powinien prowadzić ten członek zespołu projektowego, który będzie pełnił funkcję programisty wiodącego.

Technologia wytwarzania oprogramowania użytkowego z wykorzystaniem technik rodzajowych sprowadza się do tworzenia podstaw logicznych oprogramowania użytkowego na etapie projektowania koncepcyjnego, według metody PSL/PSA. Dekomponując projektowany system informatyczny do poziomu funkcji programowych uzyskuje się możliwość otrzymania około 50% kodu źródłowego oprogramowania użytkowego w języku COBOL.

ORGANIZACJA PRACY W ZESPOLE WYKORZYSTUJĄCYM TECHNIKI RODZAJOWE

Po zakończeniu procesu projektowania koncepcyjnego systemu, programista wiodący wykonuje analizę wykonanego opisu projektu, posługując się technikami rodzajowymi. Wynikiem ich działania jest biblioteka makrodefinicji zawierająca:

— opis w języku COBOL wszystkich struktur danych zdefiniowanych w projekcie w języku PSL,
— opis w języku COBOL wszystkich szkieletów programów, które funkcjonalnie zdefiniowano w konwencji języka PSL.

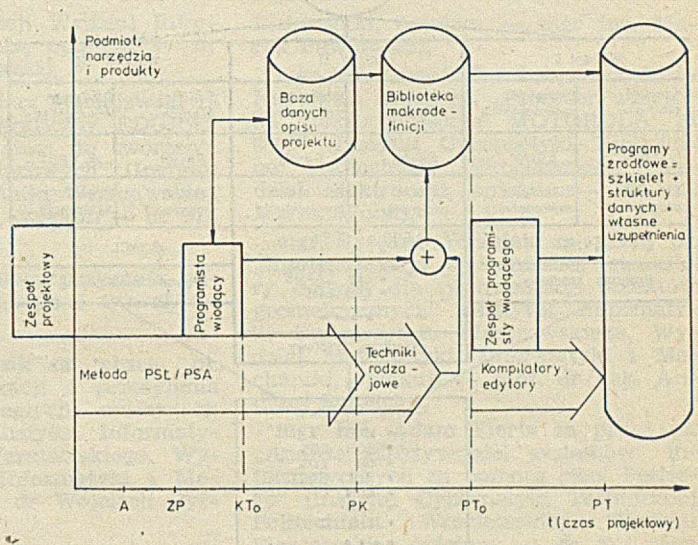
Uzyskanie takiej biblioteki makrodefinicji w sposób automatyczny z bazy danych opisu projektu jest podstawą do rozdzielenia i wydania tzw. uzupełniających zadań programowych. Członkowie zespołu programisty wiodącego mają w sposób maszynowy zdefiniowane swoje zadania programowe. Otrzymując do wykonania żądany program dysponują jego szkieletem i zbiorem definicji struktur danych, którymi wprost uzupełniają szkielet swojego programu. Ostatnim stadium uzyskania pełnego programu źródłowego jest uzupełnienie tak zdefiniowanej konstrukcji specyficznymi procedurami, których funkcje określono w szkielecie.

Wstępna analiza programu, bez potrzeby tworzenia schematu blokowego i jego uzupełnienie strukturami danych odbywa się automatycznie na poziomie korekty, kompilacji lub edycji. Zawartość każdego szkieletu programu jest wyrażona przy użyciu konstrukcji strukturalnych języka COBOL. Dlatego też końcowa postać źródłowa każdego programu jest strukturalna.

Programista użytkowy, jako członek zespołu programisty wiodącego, operuje wprost maszynową postacią swojego produktu z wyłączeniem faz poprzedzających w tradycyjnym działaniu fazę analizy. Założenia programowe zmieniają swoją postać i składają się w tej technologii z następujących elementów:

- wydruku „szkieletu" konkretnego programu,
- wydruku opisu struktur danych w systemie,
- fizycznej biblioteki makrodefinicji odzwierciedlającej elementy opisane na wydrukach do automatycznego wykorzystania.

Ogólną ideę prezentowanej technologii można przedstawić schematycznie, jak na rys. 1.



Rys. 1. Idea technologii ze wspomaganiem. Na osi odciętych przedstawiono czas projektowy z zaznaczeniem zakończenia etapów: A — analizy systemu, ZP — generowania zadania projektowego, PK — projektowania koncepcyjnego, PT — projektowania technologicznego oraz KT_0 — początku wytwarzania koncepcji technologii, PT_0 — powołania zespołu programisty wiodącego

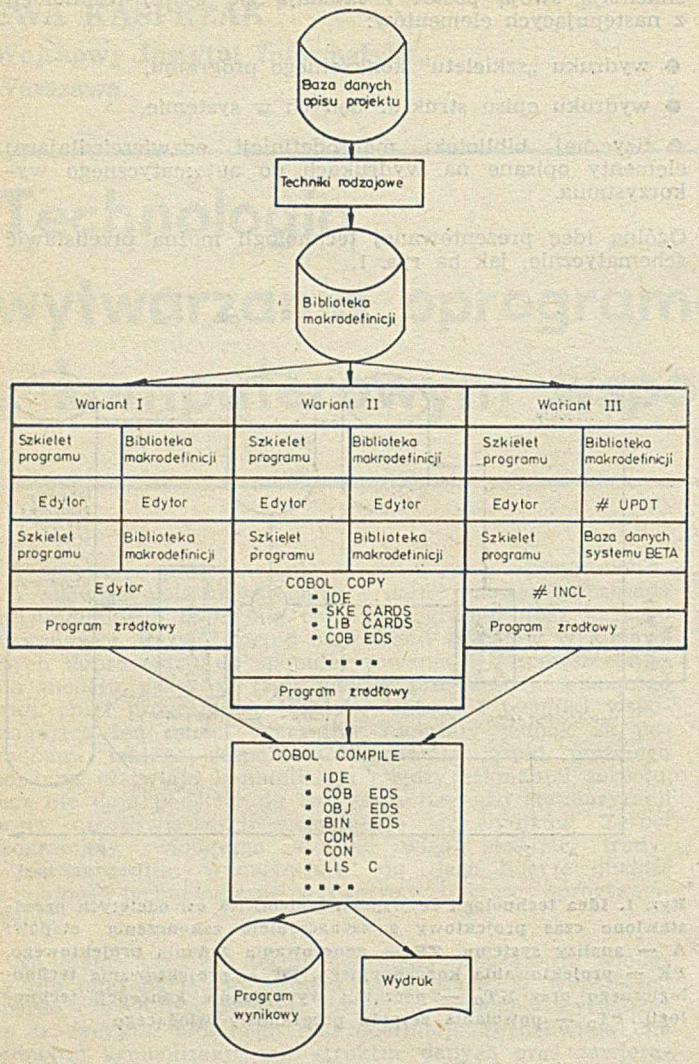
Zespół projektanta od pierwszych definicji projektowanego systemu pracuje według metody PSL/PSA. Z chwilą rozpoczęcia opracowywania koncepcji przetwarzania wydziela on ze swego składu programistę wiodącego, który definiuje elementy technologii we wspólnej bazie danych opisu projektu. Po zakończeniu prac projektowych, programista wiodący uszczegółowia opisy, posługując się tą samą bazą danych. Z chwilą zakończenia dokumentowania projektu koncepcyjnego posługuje się on technikami rodzajowymi, tworząc przy ich użyciu bibliotekę makrodefinicji, zawierającą opisy wszystkich struktur danych i szkielety występujących w systemie programów.

Zespół programisty wiodącego, wykorzystując dostępne narzędzia programowe (kompilator, edytor), tworzy programy źródłowe i wynikowe, opierając się na udostępnionej mu bibliotece makrodefinicji. Programista wiodący skupia się przede wszystkim na odbiorze i całościowym testowaniu oprogramowania użytkowego.

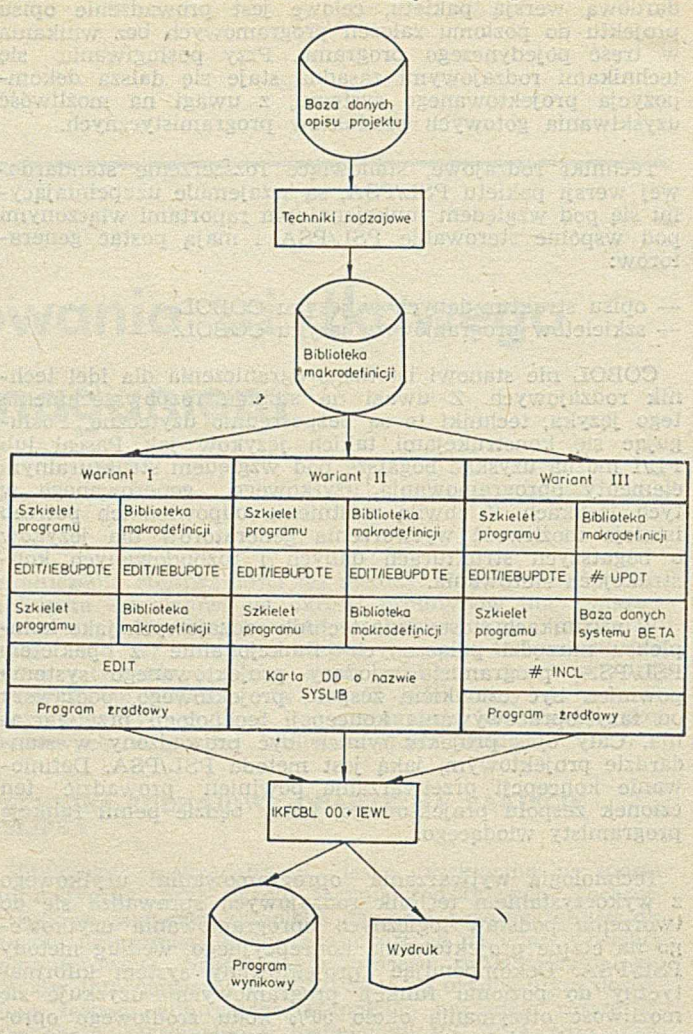
Schemat wykorzystania tego kompleksu narzędzi przedstawiono na rysunkach 2 i 3. Ilustrują one ideę posługiwania się wytworzonym przy użyciu technik rodzajowych produktem dla komputerów ODRA-1300 i R-32. Posługiwanie się wygenerowanymi elementami technologii jest podobne dla obu komputerów. W obu wypadkach istnieją trzy warianty wykorzystania wytworzonej biblioteki makrodefinicji:

- **w wariancie pierwszym** wykorzystuje się dostępny na danym komputerze edytor w celu dołączenia gotowych fragmentów kodu do programu źródłowego,
- **w wariancie drugim** stosuje się uzupełniająco fazę COPY kompilatora COBOLu w celu automatycznego dołączenia potrzebnych elementów z biblioteki makrodefinicji,

Rys. 2. Schemat operowania technologią na komputerze ODRA-1300



Rys. 3. Schemat operowania technologią na komputerze R-32



● **w wariancie trzecim** przedstawiono możliwość użycia systemu BETA wchodzącego w skład pakietu PSL/PSA, który umożliwia utworzenie autonomicznej bazy danych z biblioteki makrodefinicji i automatyczne (szybsze niż w wariancie drugim) włączanie elementów technologii do programu źródłowego.

EFEKTY STOSOWANIA TECHNIK RODZAJOWYCH

Jednoosobowe przygotowanie i wygenerowanie dostępnymi generatorami biblioteki makrodefinicji zapewnia:

● spójność logiczną i technologiczną projektu systemu z jego oprogramowaniem przez operowanie wspólną bazą opisu projektu systemu,

● jednolitą symbolizację, nazewnictwo i konstrukcję logiczną oprogramowania użytkowego,

● jednoznaczną komunikację w zespole programisty wiodącego,

● strukturalną konstrukcję programu,

● spójną technologicznie pracę, zarówno na komputerze R-32 jak i na komputerze ODRA-1300, tak w wersji wsadowej, jak i interakcyjnej, co wiąże się z przenośnością opisu projektu z jednego komputera na drugi,

● skracanie czasu projektowania technologicznego po opanowaniu tej technologii wraz z biegłym opanowaniem techniki PSL/PSA przez programistę wiodącego i zespoły projektantów,

● dyscyplinę wykonawczą oraz wyłączenie dzięki tej technologii niektórych faz produkcji oprogramowania użytkowego z indywidualnego wykonawstwa.

Istotą prezentowanej technologii produkcji oprogramowania użytkowego systemów informatycznych jest scedowanie wykonania rutynowych i powtarzalnych czynności na komputer. Komputerowe wspomaganie procesu projektowania i programowania użytkowych systemów informatycznych stymuluje wiele zmian jakościowych w modelu pracy zespołów. Po właściwym opanowaniu warsztatu prowadzi to do efektów ekonomicznych. Z uwagi na duży udział kosztów wytwarzania oprogramowania w kosztach całego przedsięwzięcia projektowego, poszukiwanie spójnego warsztatu projektanta i programisty oraz wspomaganie procesu wytwarzania oprogramowania użytkowego jest niezbędne i użyteczne. Potwierdzają to praktyczne działania związane z wykorzystaniem omawianego kompleksu narzędzi w projektowaniu użytkowych systemów informatycznych.

LITERATURA

[1] Kasprzak E., Zaskórski P.: O technikach generycznych projektowania i wytwarzania technologii systemów informacyjnych zarządzania. INFOGRYF'84

[2] Nowakowski A. red.: PSL/PSA we wspomaganiu projektowania systemów informatycznych. Politechnika Szczecińska, 1982

[3] Zaskórski P.: Komputerowe wspomaganie projektowania i wytwarzania technologii systemów informatycznych. Materiały Konferencji Naukowo-Technicznej, Rydzyna, 1983

[4] Zaskórski P. i in.: Dokumentacja eksploatacji i użytkowania systemu komputerowego wspomagania projektowania systemów informatycznych i wytwarzania ich oprogramowania użytkowego bazującego na pakiecie PSL/PSA. Wojskowy Instytut Informatyki, Warszawa, 1984.

Konkurs Polskiego Towarzystwa Informatycznego na najlepsze prace magisterskie z informatyki — rozstrzygnięty

Zakończył się II Ogólnopolski Konkurs Polskiego Towarzystwa Informatycznego na najlepsze prace magisterskie z informatyki, zorganizowany przez Dolnośląski Oddział PTI. Na posiedzeniu w dniu 16 grudnia 1985 roku Komisja Konkursowa PTI w składzie: prof. dr hab. Andrzej Blikle, doc. dr inż. Czesław Daniłowicz (przewodniczący), dr inż. Zbigniew Huzar, doc. dr hab. Maciej Sysło, dr inż. Zbigniew Szpunat (sekretarz), prof. dr hab. Władysław M. Turski i doc. dr hab. inż. Jan Zabrodzki, powzięła następujące decyzje.

Pierwszej nagrody nie przyznano.

Drugą nagrodę w wysokości 15 tys. zł otrzymała **mgr inż. Joanna Ayed** za pracę pt. „Moduł badający wydajność systemu operacyjnego George-3" (Instytut Informatyki Politechniki Warszawskiej, Wydział Elektroniki, opiekunka — mgr inż. Ilona Bluemke).

Dwie równorzędne trzecie nagrody po 10 tys. zł otrzymali: **mgr inż. Marek Krzemiński** za pracę pt. „Procesor języka QUERY BY EXAMPLE" (Zakład Systemów Informacyjnych Politechniki Wrocławskiej, Wydział Informatyki i Zarządzania, opiekun — dr inż. Bogumił Szabłowski),

mgr inż. Włodzimierz Maciejewski za pracę pt. „Edytor graficzny EDGRAF — interakcyjny system do tworzenia rysunków dwuwymiarowych" (Instytut Informatyki Politechniki Warszawskiej, Wydział Elektroniki, opiekun — dr inż. Cezary Stępień).

Komisja Konkursowa przyznała ponadto pięć wyróżnień po 5 tys. zł. Otrzymali je:

mgr Tomasz Rądzik za pracę pt. „Problemy komunikacji i przesyłania danych w równoległych maszynach synchronicznych" (Instytut Informatyki Uniwersytetu Warszawskiego, Wydział Matematyki, Informatyki i Mechaniki, opiekun — dr Wojciech Rytter),

mgr Piotr Stodółkiewicz za pracę pt. „Wznawianie programu w systemie mikrokomputerowym po wystąpieniu błędów w działaniu sprzętu lub w programie" (Instytut Informatyki Politechniki Warszawskiej, Wydział Elektroniki, opiekun — mgr inż. Marek Pawłowski),

mgr inż. Leszek Żur i mgr inż. Iwan Nepow za pracę pt. „Procesor wizyjny na mikroprocesorze MOTOROLA M 6802" (Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, Wydział Elektroniki, opiekun — dr inż. Marek Wnuk),

mgr Ryszard Woźniak za pracę pt. „Implementacja dynamicznej struktury danych dla reprezentacji obiektów geometrycznych" (Instytut Informatyki Uniwersytetu Warszawskiego, Wydział Matematyki, Informatyki i Mechaniki, opiekun — prof. dr hab. Andrzej Salwicki),

mgr inż. Adam Fierla za pracę pt. „Analiza efektywności systemów informatycznych za pomocą sieci Petriego" (Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, Wydział Elektroniki, opiekun — dr inż. Jan Magot).

Laureatom konkursu serdecznie gratulujemy!

BARBARA OSUCHOWSKA
rzecznik prasowy PTI

Szybki system operacyjny DOS

W wielu ośrodkach obliczeniowych nadal eksploatowany jest system operacyjny DOS dla komputerów IBM 360 i Riad 32. Zazwyczaj stosuje się przy tym system wspomagający POWER, który — jak wiadomo jego użytkownikom — pracuje na podstawie systemowej biblioteki programów, znajdującej się na dysku rezydentowym systemu. Powoduje to konieczność bardzo częstego dostępu do tego dysku. Ponadto, sięgnięcie do żądanych modułów jest poprzedzone dostępem do skorowidza (ang. directory) biblioteki, który znajduje się na innym cylindrze, wskutek czego następuje znaczne, również mechaniczne, obciążenie stacji dyskowej (rzędu tysięcy ruchów głowic na godzinę). Nie trzeba dodawać, że powoduje to zużycie wszystkich części tej stacji oraz znaczne straty czasu na odpowiednie ustawienie głowic.

Dzięki rozwojowi technologii pamięci półprzewodnikowych o dużej pojemności, możliwe (i stosunkowo mało kosztowne) stało się rozwiązanie tego problemu — w sposób podobny jak w systemach OS i DOS/VS. Zrealizowane zmiany w systemie operacyjnym DOS wykonano rozszerzając pamięć operacyjną co najmniej do 256 KB.

Pierwszym etapem zmian jest umieszczenie w pamięci operacyjnej skorowidza (ang. directory) biblioteki systemowej. Dzięki temu uzyskuje się widoczną poprawę w pracy systemu, polegającą na tym, że głowice stacji systemowej zachowują się o wiele spokojniej. Oszczędności czasowe są niewielkie, aczkolwiek znaczące.

Drugim etapem jest zrealizowanie przechowywania w pamięci operacyjnej najczęściej używanych modułów, takich jak: tzw. transienty systemu, Job Control, Job Accounting System, POWER itd. Do tego celu jednak niezbędna jest pamięć operacyjna o pojemności co najmniej 512 KB. Lista modułów rezydentnych jest definiowana przez użytkownika i może być zmieniona w zależności od rodzaju wykonywanych prac. Wobec programów rezydentnych nie stawia się żadnych wymagań, tak że rezydentny może być dowolny moduł, zarówno systemowy jak i użytkowy.

Dla komputerów o różnej pojemności pamięci operacyjnej można zastosować różne zestawy modułów rezydentnych, np.:

- od 1 MB — wszystkie transienty A i B systemu, Job Control i Accounting, Linkage Editor, FGP (POWER), SORT, wybrane kompilatory i programy użytkownika,
- od 512 kB — transienty A i B, Job Control i Accounting,
- od 256 kB — tylko rezydentny skorowidz biblioteki, ewentualnie wybrane transienty A i B (np. open-close, komunikacja z operatorem).

Efekty zastosowania tego rozwiązania dla pamięci 1 MB są następujące:

- wyeliminowanie średnio 5000 ruchów głowic na godzinę,
- oszczędność czasu przetwarzania średnio 30 s na pracę (ang. job)
- skrócenie czasu wykonania poszczególnych programów (np. SORT co najmniej o 40 s, LNKEDT o 15 s, kompilacja PL/I o 25 s, tłumaczenie przez asembler o 20 s),
- olbrzymie przyspieszenie przetwarzania plików indeksowo-sekwencyjnych.

Wprowadzenie opisanego rozwiązania jest możliwe dla dowolnej wersji systemu DOS i nie powoduje żadnego ograniczenia jego funkcji. Zmodyfikowany system jest na tyle elastyczny, że rozszerzone funkcje mogą być dynamicznie deaktywowane i aktywowane bez zakłócenia pracy. Jedynym mankamentem tego rozwiązania jest konieczność deaktywowania funkcji systemu w wypadku śledzenia za pomocą programu PDAID.

Opisany system działa z powodzeniem od kilku miesięcy w Fabryce Samochodów Ciężarowych w Starachowicach. **Bliższych informacji udzielają autorzy:**
dr inż. Grzegorz Czaja i mgr inż. Leszek Fiedorowicz z Ośrodka Obliczeniowego Fabryki Samochodów Ciężarowych, Starachowice, tel. 88-31 wew. 12-38.

Strukturalizacja Basica

Basic, wraz ze wszystkimi odmianami, jest jednym z najbardziej rozpowszechnionych języków programowania, mimo że wielu programistów narzeka na jego braki, a najbardziej — brak cech charakterystycznych dla języków strukturalnych. Pewne niedostatki języka można jednak skompensować tworząc bibliotekę procedur i funkcji.

Większość języków strukturalnych ma dwa rodzaje bibliotecznych programów standardowych: funkcje i procedury. W Basicu pierwsze są podprogramami obliczającymi tylko jedną wartość, drugie — więcej niż jedną lub żadnej.

W Basicu nie ma pojęcia zmiennych lokalnych ani nie dopuszcza się przekazywania do podprogramów zmiennych o różnych nazwach. Programy w Basicu muszą mieć numerowane linie, zatem wprowadzając podprogram, na skutek kolizji numerów linii, można zniszczyć poprzedni program.

Z tego powodu programiści zwykle zmieniają nazwy zmiennych i przenumerowują linie podprogramu, używając go w nowym programie, lub co gorsze — aby kod był bardziej zwarty — tworzą podprogramy na użytek tylko jednego programu. Przy odrobinie wysiłku można jednak upodobnić strukturę Basica do prawdziwych procedur i funkcji, tym samym zmniejszając rozwlekłość programów. Taką koncepcję zrealizowano w nowszych odmianach Basica, m.in. w ostatniej wersji DEC VAX BASIC, BETTER BASIC firmy Summit Technology i TRUE BASIC firmy TRUE BASIC Inc.

Jako podstawę dalszych rozważań przyjęto interpreter Microsoft BASIC i kilka bardziej popularnych odmian Basica, jak GW-BASIC, Applesoft BASIC i Commodore 64 BASIC.

Strukturalizacja wymaga opracowania szeregu reguł programowania i ich ścisłego przestrzegania. Taka dyscyplina może utrudniać programowanie w pierwszym okresie, lecz poniesiony wysiłek jest kompensowany przez stworzoną bibliotekę. Uruchamiając program można bardziej skoncentrować się na jego dopracowywaniu pod kątem użyteczności, niż na problemach technicznych, takich jak operowanie danymi lub menu. Programy napisane według tych samych reguł będą tak samo komunikowały się z otoczeniem, a całe bloki programu będą wolne od błędów już przed pierwszym jego uruchomieniem. Każdy znający taką bibliotekę podprogramów będzie z łatwością posługiwać się wykorzystującymi je programami.

Bardzo łatwe jest też dostosowanie programów do różnych komputerów, tzn. przekształcenie biblioteki.

Reguły numerowania linii

Bez przyjęcia pewnych zasad numerowania linii, dołączanie podprogramów może spowodować skasowanie programu (lub jego części) lub powoduje konieczność przejrzenia programu pod względem wolnych numerów linii, czego skutkiem jest ładowanie, przenumerowanie i zapisanie podprogramu, ponowne załadowanie programu i dołączenie podprogramu. Aby tego uniknąć, wystarczy podzielić program na logiczne części i każdej z nich przypisać odpowiednie numery linii.

Programy zawsze zaczynają się od inicjalizacji lub deklaracji. Kilka komentarzy powinno opisywać cel programu, a w następnych liniach należy umieścić deklaracje tablic i zmiennych tekstowych.

Ideałem byłoby inicjalizowanie każdej zmiennej o określonym znaczeniu w programie i dodanie komentarza objaśniającego jej rolę. Na ten cel wydaje się wystarczające zarezerwowanie linii o numerach 0-999.

Niektórzy programiści umieszczają inicjalizacje na końcu programu wywołując je instrukcją GOSUB. Taki sposób inicjalizowania czyni program nieprzejrzystym, a co ważniejsze jest niezgodny z zasadą stosowaną w większości kompilatorów, wymagają deklarowania tablic przed wywołaniem.

Następny segment powinien wywoływać podprogramy biblioteczne w odpowiedniej kolejności, co nadaje całości programu indywidualny, niepowtarzalny charakter. Na tę część można przeznaczyć linie o numerach 1000—9999.

Na podprogramy nie będące częścią biblioteki, wykorzystywane tylko w jednym programie, można przeznaczyć linie od 10 000 do 14 999.

Niektóre zastosowania wymagają użycia specjalnych podprogramów aplikacyjnych. Na przykład, program obliczania listy płac może używać podprogramu odczytywania z pliku nazwisk pracowników, a program inwentaryzowania magazynu może mieć podprogram obliczający, które materiały należy uzupełnić. Dla takich podprogramów można zarezerwować linie od 15 000—24 999.

Na końcu powinny znajdować się podprogramy systemowe — biblioteka oprogramowania, używanego w każdym programie. Dla nich należy przeznaczyć główną część obszaru programu, powiedzmy linie 25 000—65 000, które w praktyce będą bardzo szybko zajmowane.

Podprogramy systemowe można zróżnicować według spełnianych funkcji. Linie 25 000—29 999 mogą być zajęte przez podprogramy współpracy z użytkownikiem, jak np. menu; linie 30 000—34 999 można zarezerwować na obsługę plików; 35 000—59 999 na podprogramy uniwersalne, jak operowanie danymi i tekstami, a dla podprogramów obsługi błędów ON ERROR GOTO można przeznaczy linie 60 000—65 000. Powyższe zasady podsumowano w tab. 1.

Tabela 1. Zasady numerowania linii

Segment programu	Numer linii
Inicjalizacja (deklaracje)	0— 999
Program główny	1000— 9999
Podprogramy niebiblioteczne	10000—14999
Podprogramy aplikacyjne	15000—24999
Podprogramy biblioteczne	25000—65000
— komunikacji z użytkownikiem	25000—29999
— obsługi plików	30000—34999
— uniwersalne	35000—59999
— ON ERROR GOTO	60000—65000

W niektórych wersjach Basica umieszczenie podprogramów za segmentem inicjalizacji powoduje szybsze wykonywanie programu. Program napisany w Basicu jest przechowywany w pamięci jako lista linii, z których każda wskazuje swój następnik. Wynika stąd, że linie znajdujące się bliżej początku programu są odszukiwane szybciej niż linie z końca. Co prawda, w powszechnie przyjętej wersji GW-BASIC firmy Microsoft poszukuje się linii w ten sposób tylko przy pierwszym przywołaniu, później wykonując skoki bezpośrednie, lecz oszczędność czasu jest tak minimalna, że z całym spokojem można ją zignorować.

Używanie systemu o zaproponowanej strukturze nie odbywa się bez pewnych dodatkowych warunków. Po utworzeniu biblioteki należy sprawdzić, czy jakieś podprogramy nie zachodzą na siebie. Jeśli powstaje konieczność uzupełnienia lub poprawienia podprogramu, to nowe linie należy raczej wprowadzać między już istniejące niż „przedłużać" (rozciągać) cały podprogram. Bardzo użyteczne jest numerowanie linii za skokiem 10. Proponowane reguły nie powinny jednak być traktowane zbyt sztywno. Należy przewidzieć, ile linii będą wymagały poszczególne segmenty programu i w zależności od tego przydzielać im obszary. Przykładowo, najwięcej miejsca przeznacza się na podprogramy biblioteczne. Choć tylko niektóre z nich będą używane w każdym programie, wszystkie muszą spełniać wymagania stawiane bibliotece.

Niektórzy programiści nigdy nie zaczynają podprogramów od komentarzy (REM), ponieważ wiele programów kompresji kasuje je zostawiając odpowiednie linie nieokreślone. Oczywiście, używajac bardziej wyrafinowanych pakietów można stosować komentarze unikając kłopotów. Do takich pakietów należy zaliczyć: Basic Development System, firmy SoftTool — dla mikrokomputera IBM PC; GPLE, firmy Beagle Bros — dla Apple i Snapp Utilities — dla TRS-80. Wszystkie trzy pakiety zawierają inne podprogramy użytkowe ułatwiające programowanie. Inny sposób unikania wspomnianej niedogodności polega na umieszczeniu komentarzy przed podprogramem (np. instrukcja REM może zajmować linie 5196—5199, a sam podprogram — zaczynać się od 5200).

Nazwy

W wielu odmianach Basica nazwy obiektów nie mogą być dłuższe od ośmiu znaków, co stanowi dużą przeszkodę w stosowaniu nazw opisowych, wyjaśniających użycie podprogramów. Z tego powodu należy przyjąć konwencję, według której nazwa podprogramu zaczyna się dwuznakowym przedrostkiem, np. PS dla podprogramów systemowych. Podprogramy specjalne mogą mieć przedrostki będące skrótem ich przeznaczenia, np. LP dla listy płac. Użycie takiej metody umożliwi szybkie zidentyfikowanie rodzaju podprogramu i liczby jego linii. Dodatkowa dokumentacja powinna dokładnie opisywać działanie podprogramu.

Takie reguły tworzenia nazw zmiennych zabezpieczają przed nieoczekiwanymi efektami ubocznymi, występującymi wtedy, gdy program główny i podprogram używają tej samej zmiennej do różnych celów. Przykładowo, gdy program główny wykorzystuje zmienną PR do rejestracji wpłat, a podprogram używa jej jako wskaźnika stanu drukarki, to jest oczywiste, że wywołanie podprogramu zmienia wartość zmiennej PR.

W celu uniknięcia takich kłopotów można symulować istnienie zmiennych lokalnych przez przypisanie grup zmiennych logicznym częściom programu. Przykładowo, programowi głównemu i podprogramom nie będących częścią biblioteki można przydzielić zmienne zaczynające się literami A-H, pętlom programu głównego — zmienne zaczynające się od I-J (podobnie jak w FORTRANIE), K-P podprogramom użytkowym, a podprogramom systemowym — Q-Z. W tab. 2 przedstawiono podsumowanie tych reguł.

Można pójść jeszcze dalej i podzielić zmienne na mniejsze podgrupy. W każdym segmencie programu występują dwa rodzaje zmiennych: tymczasowe, używane w obliczeniach pośrednich i później nie wykorzystywane, oraz zmienne permanentne, które mają znaczenie w całym programie lub w określonym zastosowaniu. Zmienne tymczasowe mogą zaczynać się literami A-D, a permanentne literami E-H. Podobnie jak numery linii, zmienne można podzielić w zależności od ich roli w programie.

Przy wywołaniu podprogramu należy ustawić zmienne lokalne na wartości, które trzeba przekazać. Po powrocie z podprogramu, wartości przeznaczone do zapamiętania należy jak najszybciej zapisać — zanim wywoła się inny podprogram. W przeciwnym wypadku nowo wywołany podprogram mógłby je zmienić.

Nazwy opisowe zaleca się stosować tylko w programie głównym, podprogramach niebibliotecznych i aplikacyjnych, lecz tylko dla zmiennych permanentnych. Używanie takich nazw w podprogramach bibliotecznych pozbawiłoby je zalet uniwersalności (wyjątkiem są nazwy **true** i **false** mające zawsze tę samą definicję). Należy być świadomym potencjalnych trudności w tych odmianach Basica, w których znaczące są tylko dwa pierwsze znaki nazwy zmiennej.

Zakładanie biblioteki

Każdy, kto chce z powodzeniem stosować powyższe zalecenia, powinien je starannie zapisać i przekazać wszystkim potencjalnym użytkownikom biblioteki. Następnym krokiem po ustaleniu reguł programowania, powinno być utworzenie szkieletu programu na użytek każdego nowego projektu. W jego pierwszej części należy ustalić kolejność inicjalizacji, aby była zgodna z logiką i ułatwiała kompilację (tab. 3). Druga część programu powinna zawierać komentarze opisujące podział ról, np.:

1000 REM PROGRAM GŁÓWNY 1000—9999

Przed umieszczeniem podprogramu w bibliotece należy odpowiedzieć na pytania, czy będzie używany w wielu programach i czy jest dostatecznie uniwersalny bez utraty funkcjonalności? Jeśli odpowiedź na obydwa pytania jest twierdząca, to wybrany podprogram nadaje się do biblioteki. Przykładowymi podprogramami tego rodzaju są: **"Press RETURN to Continue"**, usuwanie początkowych i końcowych odstępów z napisu, obliczanie wartości numerycznej napisu w celu sprawdzenia, czy jest poprawną liczbą itd. Bardziej rozbudowane podprogramy standardowe mogą zawierać menu współpracujące z kursorem lub klawiszem RETURN.

Programy powinny być tworzone z odrobiną wyobraźni. Przykładowo, dobry podprogram wejściowy do pobierania tekstu o określonej długości z klawiatury, powinien umożliwiać używanie klawiszy funkcyjnych, wyświetlając informacje: **"Naciśnij F1 w celu przejścia do tyłu, F2 — do przodu, F3 — zakończenia".**

Tabela 2. Zasady tworzenia nazw zmiennych

Segment programu	Nazwa zmiennej
Program główny	A-H, I-J dla pętli FOR... ← Next
— tymczasowe	A-D
— pernamentne	E-H, nazwy opisowe
Podprogramy niebiblioteczne	A-H
— tymczasowe	A-D
— permanentne	E-H
Programy aplikacyjne	K-P
— tymczasowe	K-M
— permanentne	N-P
Podprogramy	Q-Z
— tymczasowe	Q-R
— permenentne dla procedur komunikacji	S-T
— permenentne dla obsługi plików	U-V
— permenentne dla podprogramów uniwersalnych	W-X
— permenentne dla obsługi błędów ERROR GOTO	Y-Z
— zmienne systemowe (true, false, itp.)	

Tabela. 3. Zalecana postać inicjalizacji (zdecydowanie złą praktyką jest używanie kilku instrukcji w jednej linii, jak w linii 120 — przyp. red.)

```
— Uwagi opisujące przeznaczenie programu
— DEFINT, DEFSTR, DEFSNG, DEFDBL, służące do określania typu zmiennych
— DIM, deklarowanie tablic

COMMON, przekazywanie zmiennych między programami
ON ERROR GOTO
DEF FN, definiowanie funkcji
Deklarowanie zmiennych programu
100 REM 100-199 Deklaracja zmiennych programu
110 REM Data, nazwisko użytkownika, liczba zmian
120 E0S = DATE$(0):E1$ = "":E0 = 0
```

Tabela 4. Dokumentacja procedury

```
Nazwa procedury   = CT35510
Numery linii      = 35510-35550
Przeznaczenie     = Porównuje ciąg znaków z listą ważnych haseł i udostępnia wynik
Zmienne wejściowe = Q$ = ciąg znaków
Zmienne wyjściowe = Q0 = Numer hasła (zero, jeśli dane wejściowe są niewłaściwe)

Uwaga. Podprogram wymaga obecności podprogramu CT24340 (pobranie znaków z klawiatury) w tym samym programie.
```

Konstrukcja DEF FN pozwala na definiowanie funkcji, podobnie jak w językach strukturalnych. Choć wiele wersji Basica ogranicza użycie tej konstrukcji do wyrażeń o długości jednego wiersza, nawet wtedy można tworzyć funkcje dość złożone.

Tworzone podprogramy należy jak najszerzej dokumentować zarówno w treści programu, jak i na papierze. Pierwsze trzy linie każdego podprogramu powinny opisywać jego przeznaczenie oraz wymagane i udostępniane zmienne. Natomiast dokumentacja pisemna powinna umożliwiać zidentyfikowanie każdego podprogramu i dokładnie wyjaśniać jego działanie i użycie. W tab. 4 przedstawiono zalecaną postać takiej dokumentacji.

Biblioteka podprogramów będąca narzędziem codziennej pracy pozwala programiście skupić się na problemie, a nie na technikach jego rozwiązywania. Czytelnicy zainteresowani dalszym rozwijaniem tej koncepcji mogą utworzyć bibliotekę podprogramów usługowych, przeznaczonych do wykonywania większych zadań, jak na przykład zarządzanie plikami. W takim wypadku należy zdecydować się, jakimi zmiennymi opisać plik i jego pola, napisać podprogram odczytywania deskryptorów i całego pliku do pamięci, a następnie — program główny.

Opracował: MARIUSZ KUC
na podstawie: BYTE, no. 6
Vol. 10, 1985

Kto jest kim w IFIP

George Glaser (USA)

Pan George Glaser (USA) jest pierwszym wiceprezesem IFIP. W 1980 roku został mianowany po raz pierwszy delegatem Stanów Zjednoczonych do IFIP, w 1981 roku wybrany członkiem zarządu, a w 1982 roku powołany na swoje obecne stanowisko.

Karierę zawodową rozpoczął mając 17 lat, jako robotnik niewykwalifikowany w stalowni. Niższy stopień naukowy inżyniera elektryka uzyskał na Uniwersytecie Notre Dame, a następnie studiował administrację przedsiębiorstw na Uniwersytecie w Nowym Meksyku. Po skończeniu studiów pracował, m.in. jako konstruktor elektryk, programista systemów, konserwator urządzeń, planista produkcji i kierownik produkcji (zakładu cyfrowych taśm magnetycznych).

W 1961 roku związał się z międzynarodową firmą konsultacyjną, w której pracował jako doradca w różnych gałęziach przemysłu. W 1973 roku zaczął pracę na własną rękę. Cztery lata później został jednym z założycieli, prezesem i szefem produkcji dostawcy systemów rozpoznawania mowy. W 1981 roku zarejestrował swoją firmę konsultacyjną, jako George Glaser, Inc. Większość jego prac dla dużych przedsiębiorstw, w ostatnich latach, ograniczała się do wykonywania projektów systemów zarządzania, zawierających planowanie i konstruowanie urządzeń i systemów oprogramowania.

George Glaser napisał wiele artykułów w profesjonalnych czasopismach i publikacji handlowych (miał więcej wystąpień jako mówca) na takie tematy jak: kierowanie projektami, modelowanie ekonomiczne, zarządzanie przetwarzaniem danych i rozpoznawanie mowy.

George Glaser działa w stowarzyszeniach zawodowych od 25 lat. Zaczął jako pracownik obsługi technicznej słynnej Ogólnoamerykańskiej Konferencji Komputerowej (ang. Joint Computer Conference) w 1960 roku i po sześciu latach został przewodniczącym Komitetu Organizacyjnego. Przez dwie kadencje był prezydentem Amerykańskiej Federacji Towarzystw Przetwarzania Informacji AFIPS (ang. American Federation of Information Processing Society i przewodniczącym rady dyrektorów. Brał aktywny udział w pracach Association for Computing Machinerg (ACM) jako skarbnik, nieetatowy członek rady i przewodniczący grupy przetwarzania danych w przedsiębiorstwach.

Jest ojcem dwóch synów i córki oraz dziadkiem dwóch wnuczek. Mieszka w Los Altos w Kalifornii ze swoją żoną Karen Duncan.

MK — na podstawie IFIP Bulletin

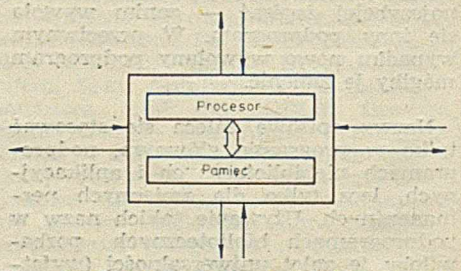
Transputery — przyszłość techniki komputerowej?

Transputer jest małym, lecz kompletnym procesorem, który użyty jako blok konstrukcyjny pozwala tworzyć sieci obliczeniowe o dużych możliwościach. Językiem programowania transputerów jest Occam, a ich moc obliczeniowa pozwala wykonywać ponad milion operacji na sekundę (mips, ang. million instructions per second), co jest porównywalne z mocą obliczeniową najnowszych japońskich komputerów zaliczonych do piątej generacji.

Architektura transputera opiera się na klasycznej, stosowanej w pierwszych komputerach, koncepcji von Neumanna. Komputer von Neumanna ma jeden procesor centralny połączony bezpośrednio jedną magistralą z pamięcią. Konstrukcja ta była w miarę upływu lat modyfikowana w kierunku zwiększenia mocy obliczeniowej, przez efektywniejszą współpracę z pamięcią operacyjną. Przykładem takiej modyfikacji są układy bezpośredniego dostępu do pamięci (ang. direct memory access — DMA).

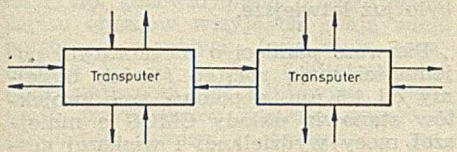
Wraz z ewolucją architektury rozwijały się języki programowania. Możliwości programowe były jednak zawsze ograniczone architekturą komputerów, a szybkość obliczeń — pojedynczą magistralą między procesorem a pamięcią. Ta istotna wada architektury klasycznej stała się główną zasadą transputerów, prowadząc do zwiększenia mocy obliczeniowej przez powielenie układu podstawowego w strukturę obliczeniową.

Transputer jest więc komputerem jednoukładowym, posiadającym cztery sprzęgi wejścia-wyjścia pozwalające łączyć go z innymi transputerami (rys. 1). Systemy mogą mieć różną strukturę. Najprostszy układ składa się z dwu transputerów (rys. 2). Taki system jest traktowany jak układ o procesorze głównym i koprocesorze.



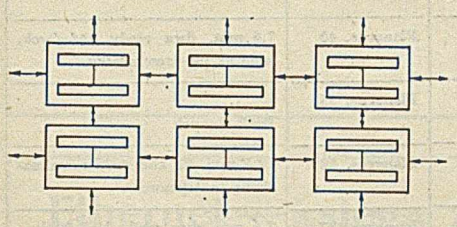
Rys. 1. Transputer o architekturze von Neumanna zaopatrzony w cztery sprzęgi wejścia-wyjścia

Układy o podziale funkcjonalnym, tzn. takie, w których każdy transputer pełni określone funkcje, mogą mieć różnorodne struktury połączeń. W przykładowej strukturze, tzw. tablicy transputerów, przedstawionej na rys. 3, każdy transputer wykonuje w danej chwili tylko jedno zadanie używając wyłącznie swojej pamięci lokalnej. Połączenie z innymi transputerami pozwala na koordynację realizowanych zadań. Sieci transputerowe mogą być też niesymetryczne lub toroidalne, symulujące tzw. sieć nieskończoną o podziale funkcji.



Rys. 2. Prosty system transputerowy

W praktyce, jako transputerów używa się komputerów jednoukładowych IMS T424 i IMS T222. Są to komputery 32- oraz 16-bitowe. Każdy z nich ma 4 KB pamięci wewnętrznej RAM oraz cztery sprzęgi wejścia-wyjścia. Do procesora T424 można dołączyć 4 GB pamięci zewnętrznej, a do procesora T222 — 64 KB. Szybkość obliczeniowa obu procesorów wynosi od 5 do 10 mips.



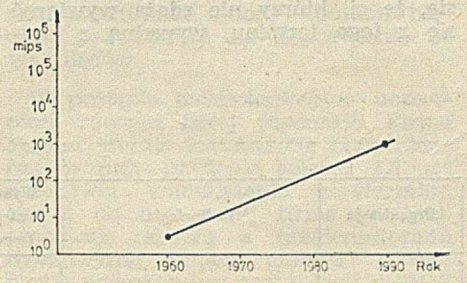
Rys. 3. Tablica transputerów

Z punktu widzenia oprogramowania każdy transputer jest implementowany jako proces języka Occam, a każde połączenie dwu transputerów stanowi dwuprzepustowy kanał. Obliczenia są interpretowane jako realizacja procesów mieszanych — sekwencyjnych i współbieżnych.

Procesor każdego transputera jest małą, sprzętową cegiełką w implementacji procesora globalnego, opisanego programowo. Oprogramowanie „czuwa" też nad przesyłaniem informacji oraz uaktywnia lub wyłącza procesory transputerów, zgodnie z zapotrzebowaniem. Możliwe są dwa tryby pracy: praca przy wyższym priorytecie, prowadząca do minimalizacji czasu odpowiedzi układu transputerowego na zgłoszenie zewnętrzne, oraz praca z priorytetem niższym — gdzie zadania są rozdzielane, zgodnie z kolejnością zgłoszeń, między poszczególne bezpośrednio połączone transputery. Zegar systemu jest realizowany programowo.

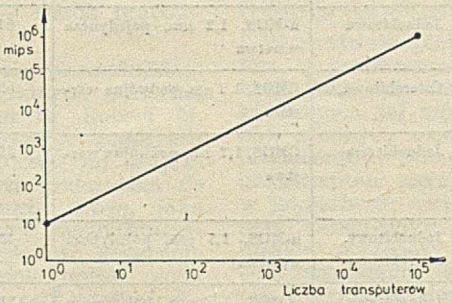
W przytoczonym niżej przykładzie przedstawiono, jak programowane są w Occamie dwa procesy składające się na działanie kanału. Program opisuje proces wysyłania tysiąca jednostek informacji oraz proces przyjęcia tysiąca tych jednostek. Wbudowany do oprogramowania zegar notuje łączny czas przesyłania informacji.

```
CHAN c:
VAR MPPS, StartTime, EndTime,
ElapsedTime:
SEQ
TIME ? StartTime
PAR
SEQi = [O FOR 1000] c!0
VAR x:
SEQ i = [O FOR 1000] c?x
TIME ? EndTime
ElapsedTime := EndTime — Start-
Time
MPPS := (1 000 000/ElapsedTime) x
x 1000
```



Rys. 4. Wzrost mocy obliczeniowej systemów konwencjonalnych

Duża szybkość obliczeniowa systemów transputerowych powoduje, że są one stosowane coraz częściej do budowy superkomputerów. Okazuje się bowiem, że rozbudowa sieci transputerowych może spowodować o wiele większy wzrost mocy obliczeniowej niż ulepszanie tzw. systemów konwencjonalnych. Na rys. 4 przedstawiono wzrost mocy obliczeniowej systemów konwencjonalnych, wyznaczony w funkcji kolejnych lat, a na rys. 5 wzrost mocy obliczeniowej systemów transputerowych w zależności od liczby transputerów tworzących te systemy.



Rys. 5. Wzrost mocy obliczeniowej systemów transputerowych

Czy sieci transputerowe będą podstawowym elementem przyszłych komputerów? Na to pytanie nie można jednak jeszcze udzielić odpowiedzi.

Oprac. TW
na podstawie Byte, May 1985

Sytuacja na rynku elementów półprzewodnikowych

Nadmierne nagromadzenie zapasów, zarówno u wytwórców, jak i większych odbiorców, doprowadziło do recesji w 1985 r. Według World Semiconductor Trade Statistics wpływy ze sprzedaży elementów spadły w USA o 28,7%, tzn. 25,9 mld dolarów. Odpowiednie liczby dla Europy i Japonii są mniejsze i wynoszą 4,5 i 6,4%. Perspektywy na 1986 rok są korzystniejsze i wpływy te, według tego samego źródła, powinny osiągnąć 25,4 mld dolarów (wzrost o 18%). Dalsze przewidywania mówią o 38,4 mld dolarów w 1988 roku.

Na razie jednak firmy półprzewodnikowe ponoszą znaczne straty, a trzeci kwartał 1985 roku był najgorszy. Np. firma National Semiconductor wykazała 44,3 mln dolarów strat w pierwszym kwartale i oceniła na 50—60 mln straty w trzecim. Przewidywano spadek sprzedaży kostek w trzecim kwartale o ok. 8—9% zarówno w USA, jak i na świecie, a był to piąty kolejny kwartał spadku. Natomiast czwarty kwartał 1985 roku według tych samych przewidywań miał przynieść 3—4% wzrostu. Już w trzecim kwartale zaobserwowano pewien wzrost zamówień, zwłaszcza od małych odbiorców, którzy nie mogą tworzyć zapasów.

Jedną z przyczyn takiej sytuacji są nadmierne moce produkcyjne aktualnie wykorzystywane w 60% (rocznie przybywa nadal 20% nowych fabryk). W rezultacie spadają ceny elementów. W porównaniu z najwyższymi cenami, jakie notowano w 1984 roku, spadły one średnio o 22,1%, przy czym ceny pamięci spadły o 31%, układów logicznych o 12%, a układów analogowych utrzymały się na nie zmienionym poziomie.

(JR)

OŚWIADCZENIE

Celem złagodzenia nieporozumień, oświadczam, że do prac nad systemami automatycznej syntezy programu pracy robotów i obrabiarek wykorzystałem II UW PROLOG, do tworzenia jego wersji tzw. PC-PROLOGu. Informacja o tym nie we wszystkich publikacjach została wyraźnie podana.

dr inż. TADEUSZ SZUBA
Instytut Informatyki AGH Kraków

Megabitowe kostki pamięciowe

Przed dwoma laty wejście na rynek kostek pamięciowych o pojemności 256 Kb współzawodniczyło około dziesięciu firm. Rozstęp ścieżek wynosił wówczas 2 μm.

Obecnie, mimo że firmy półprzewodnikowe ponoszą straty, liczba firm podejmujących produkcję kostek o czterokrotnie większej pojemności podwoiła się, choć ich technologia jest jeszcze bardziej skomplikowana (rozstęp 1 μm). Zapowiada to ogromny rozwój rynku pamięci półprzewodnikowych. Według analityków z Dataquest Inc. wielkość wpływów ze sprzedaży kostek magabitowych w 1985 roku wynosiła zaledwie 3 mln dolarów, ale w bieżącym roku ma wynosić już 260 mln dolarów. Średnia cena kostki szacowana jest (w 1986 r.) na 52 dolary, co oznacza wielkość produkcji równą 5 milionów kostek. Natomiast na rok 1990 przewiduje się sprzedaż 1,4 mld kostek pamięci półprzewodnikowych o łącznej wartości 3,5 mld dolarów.

Wśród potencjalnych producentów jest wiele firm japońskich, ale szereg firm amerykańskich, jak: Mostek, Motorola i Intel przygotowuje bardzo udane projekty w technologii CMOS, które mogą przynieść im sukcesy w bliskiej przyszłości. We współzawodnictwie biorą udział także firmy koreańskie i europejskie. Prawie wszystkie firmy planują dostarczenie próbnych partii do połowy 1986 r. Ocenia się, że ci, którzy nie zdążą wywiązać się z tego terminu, wypadną z konkurencji. Dwie firmy AT&T i Toshiba oferują próbne partie od jesieni 1985 r., a Hitachi, Fujitsu, Mitsubishi i Texas Instruments są prawie na tym samym etapie. Firma IBM wytwarza kostki na własny użytek, a firma Oki planowała podjęcie produkcji od połowy października 1985 r.

Uzyskanie kostki pamięci o pojemności miliona bitów wiąże się z pokonaniem poważnej bariery technicznej. Projektanci napotykają podobny lub nawet większy stopień trudności, aniżeli przy skoku z 16 Kb na 64 Kb pojemności dokonanym na początku obecnej dekady. Przejście na 256 Kb było już łatwiejsze.

Pierwsze pamięci o megabitowej pojemności mają rozstęp ścieżek mniejszy od 1,5 μm, a połowę tych projektów stanowią układy CMOS o mniejszej mocy wydzielanej i większej gęstości upakowania. Podobne właściwości uzyskuje się również dzięki zastosowaniu drugiej warstwy metalicznej, co

Rozwój magabitowych kostek pamięciowych

Firma	Stan zaawansowania	Organizacja	Technologia	Wymiar matrycy (mm²)	Rodzaj i pojemność (ff) kondensatorów	Obudowa
Advanced Micro Devices	Projekt końcowy, próbne partie w II-III kw. 1985	Jednobitowa, czterobitowa	CMOS, 1,2 μm, podwójna warstwa metaliczna	45,2	Rowkowe	7,6 mm, dwa rzędy końcówek, montaż powierzchniowy
AT&T	Produkcja w końcu 1985	Jednobitowa, czterobitowa	CMOS, 1,3-1 μm, pojedyncza warstwa metaliczna	44,6	Planarne, 40	7,6 mm, dwa rzędy końcówek, montaż powierzchniowy
Fujitsu	Próbne partie	Jednobitowa	n-MOS, 1,8 μm, nakładane komórki, pojedyncza warstwa	54,2	Planarne, 55	Brak danych
IBM	Wytwarzanie	Jednobitowa, dwubitowa,	SAMOS, brak danych	58,1	Planarne, 43	12,2 mm, 23 końcówki rozmieszczone matrycowo
		czterobitowa	n-MOS, 1,3 μm, podwójna warstwa	48,4	Planarne, 37	7,6 mm, dwa rzędy końcówek
Intel	Pierwsza realizacja	Czterobitowa	CMOS, 1μm, podwójna warstwa	52,9	Planarne, 48	Brak danych
Micron Technology	Realizacja w końcu 1985, próbne partie na początku 1986	Jednobitowa	n-MOS, 1,5 μm, pojedyncza warstwa	71	Planarne, 65	7,6 mm, 18 końcówek, ceramiczna
		20 różnych	CMOS, 1,5 μm, pojedyncza warstwa			Plastikowe, końcówki wyginane na przemian
Mitsubishi	Próbne partie dla odbiorców wiosną 1986	Jednobitowa	n-MOS, 1,2 μm, pojedyncza warstwa	64,5	Planarne, 45	7,6 mm, dwa rzędy po 9 końcówek
Mostek	Końcowe próby, próbna partia na początku 1986	Czterobitowa,	CMOS, 1,2 μm, podwójna warstwa	68,4	Planarne, 33	7,6 mm, dwa rzędy po 10 końcówek, montaż powierzchniowy
Motorola	Projekt końcowy, próbne partie w I kwartale 1986	Jednobitowa	CMOS, 1,2 μm, podwójna warstwa	51,6	Plan rne, 50	7,5 mm, o małym obrysie, dwa rzędu końcówek typu J, montaż powierzchniowy
NEC	Brak danych	Jednobitowa, czterobitowa	n-MOS, 1,5 μm, pojedyncza warstwa	43,2	Rowkowe, 60	16 końcówek
Texas Instruments	Próby techniczne, próbna partia dla odbiorców w I kwartale 1986	Jednobitowa, czterobitowa	n-MOS, 1,5 μm, pojedyncza warstwa	51,9	Rowkowe, 50	7,6 mm, o małym obrysie, dwa rzędy końcówek typu J
Toshiba (Siemens)	Końcowe próby, próbne partie w październiku 1985	Jednobitowa	n-MOS, 1,2 μm, podwójna warstwa	63,3	Planarne, 42	2,6 mm, dwa rzędy po 9 końcówek
		Jednobitowa, czterobitowa	CMOS, 1,2 μm, podwójna warstwa	62,6		
Vitelic	Projekt końcowy, próbne partie w pierwszej połowie 1986	czterobitowa Jednobitowa	CMOS, 1,2 μm, pojedyncza warstwa	51,6	Płaskie, 35	7,6 mm, dwa rzędy końcówek

prawdopodobnie będzie konieczne przy konstrukcjach czteromegabitowych.

Najszybsze z pierwszych modeli megabitowych mają czas dostępu poniżej 100 ns, a w pewnych rozwiązaniach statycznych parametr ten osiąga nawet wartość 25—30 ns. Większość tych układów ma kształt standardowych wydłużonych kostek o szerokości 0,3 cala (7,6 mm) z dwoma rzędami końcówek.

Ze względu na różne możliwości zastosowań projektuje się kostki o różnej organizacji informacji. Niektóre z nich, np. przewidywane do cyfrowego zapisu obrazów wizyjnych, mają wiele portów szeregowych. Inne mają organizację 1-, 2-, 4- lub 8-bitową, przy czym najwięcej jest 4-bitowych. Większość producentów planuje rozpocząć produkcję pamięci o organizacji jednobitowej, stosowanych w dużych systemach komputerowych. Niektórzy producenci zapowiadają rychłe wprowadzenie kostek 4-bitowych. Przykładowo, firma Mostek planuje rozpoczęcie w 1986 roku produkcji pamięci o organizacji 256 Kx4 bity, ale zapowiedziane wcześniej masowe dostawy kostek tej firmy o mniejszej pojemności opóźniają się. Firma United Technologies zapowiedziała produkcję kostek MK44E6 o organizacji 256Kx4 bity i MK48E6 o organizacji 128Kx8 bitów, obie w technologii MOS z rozstępem ścieżek 1,2 μm. Niektóre firmy wprowadzają do komórek pamięciowych tzw. pojemności rowkowe (ang. trench) przez kształtowanie powierzchni płytki krzemowej, co zmniejsza powierzchnię planarną, aby utrzymać stabilny ładunek. Pojemności odpowiednich kondensatorów są rzędu dziesiątków femtofaradów (10^{-15} F). Takie rozwiązanie opracowała m.in. firma Texas Instruments. Natomiast firma Motorola, również na początku 1986 roku, przedstawiła jednomegabitową pamięć CMOS o dwu warstwach metalicznych. Firma Micron Technology opracowuje 3 projekty: pierwszy — n-MOS, opierając się na produkowanych obecnie kostkach o pojemności 256 Kb i rozstępie ścieżek 1,5 μm, a dwa inne zupełnie nowe. Firma ta wprowadza też inne rozwiązanie wyprowadzenia końcówek, polegające na ich wygięciu na przemian do dwóch rzędów (ang. Zigzag Dual In-line). Firma zamierza również zastosować przyjęte w kostkach o mniejszej pojemności układy korekcji błędów, pozwalające na poprawienie 32 tys. błędów.

W postępie technologicznym uczestniczą również firmy japońskie. Firma Fujitsu stosuje konstrukcję przestrzenną, co przy rozstępie ścieżek 1,8 μm umożliwia zmniejszenie powierzchni kostki do 54,2 mm²; firma NEC opracowała kostkę z kondensatorami rowkowymi; Hitachi oferuje próbne partie wykonane w technologii n-MOS o rozstępie ścieżek 1,3 μm i strukturze jednobitowej; Mitsubishi — stosunkowo dużą (64,5 mm²) matrycę o rozstępie 1,2 μm. Toshiba sprzedała licencję firmie Siemens, obejmującą zarówno rozwiązanie w technologii n-MOS jak i CMOS. Pierwsza konstrukcja miała być w próbnej sprzedaży od października 1985 r., a od kwietnia 1986 r. przewidywano produkcję 100 tys. kostek miesięcznie.

Spośród firm amerykańskich należałoby wymienić jeszcze Advanced Micro Devices, która ma w końcowej fazie projektowania magabitową kostkę dynamiczną CMOS o rozstępie ścieżek 1,2 μm i strukturze 4-bitowej. Firma ta planuje rozpoczęcie produkcji wieloportowej kostki wizyjnej w pierwszej połowie 1987 roku. Firma AT&T sprzedaje już próbne partie po 100 dolarów za sztukę, a rozpoczęcie masowej produkcji przewidywano na początek 1986 roku, zapowiadając wersję o pojemności 256 Kx4 bity. W projektach tych wykorzystano kondensatory planarne. Nową firmą w tej branży jest Vitelic Corp. z San Jose, w Kalifornii, która zapowiada wprowadzenie w pierwszej połowie 1986 roku próbnych partii kostki megabitowej o jednobitowej organizacji, a po trzech miesiącach kostkę 4-bitową. W drugiej połowie roku oczekiwane są kostki wizyjne tej firmy.

Opracował: JAN RYŻKO
na podstawie Electronics, nr 39/85

Recenzje

Ethernet — księga źródłowa

Księga źródłowa Etherneta [1]) jest poprawionym i rozszerzonym wydaniem podręcznika „The Ethernet Handbook", którego dwa poprzednie wydania, zawierające wszelkie dostępne informacje na temat lokalnej sieci komputerowej Ethernet, cieszyły się dużym powodzeniem w środowiskach przemysłowych i akademickich, a także — wśród pracowników instytucji rządowych.

Księga źródłowa Etherneta jest w istocie przewodnikiem po wyrobach i producentach urządzeń zgodnych z normą sieci Ethernet. Wstęp do książki napisał dr Robert M. Metcalfe, główny twórca tej sieci. Dokonując przeglądu właściwości sieci wyjaśnił on przebieg procesu jej projektowania, w którym zerwano z tradycyjnym podejściem do zagadnień obliczeniowych. Przedstawił także drogę, jaką przeszła specyfikacja sieci przez organizacje normalizacyjne, zanim stała się normą, i omówił przyszłość Etherneta jako lokalnej sieci komputerowej. Następnie opisał plany rozwojowe sieci Ethernet dla komputerów osobistych i dokonał przeglądu istniejących opracowań. Na zakończenie tego wprowadzenia autor sformułował pewne prognozy dotyczące rozwoju sieci Ethernet, przemysłu wytwarzającego urządzenia dla sieci lokalnych i — rozwiązania najważniejszych kwestii spornych wynikających z normy.

Pierwszy rozdział książki zawiera przedruk kilku artykułów wyznaczających najważniejsze daty w historii rozwoju tej sieci [2,3]). Większość z nich została opublikowana w znanych czasopismach i charakteryzowała podstawowe właściwości Etherneta.

W drugim rozdziale przedstawiono normę IEEE Std 802-3, stanowiącą specyfikację sieci opartej na zasadach Etherneta, a w trzecim opisano możliwości połączenia komputerów osobistych z siecią, przedstawiając jedno z przykładowych rozwiązań.

Rozdział czwarty zawiera szczegółowy opis kilku instalacji sieci Ethernet, m.in. na Uniwersytecie stanu Utah w Salt Lake City (USA) i w Standard Oil Company w Kalifornii. Na tej podstawie czytelnik uzyska dość dobry obraz praktycznych rozwiązań sieci i jej zastosowań.

W rozdziale piątym umieszczono skorowidz do informatora o producentach urządzeń sieci Ethernet i ich wyrobach. Sam informator zaś, będący zasadniczą częścią książki, stanowi jej szósty rozdział. Zawarto w nim opis wyrobów zgodnych z normą, wytwarzanych przez ok. 70 różnych producentów. Stanowi on nieocenione źródło informacji dla projektantów sieci, inżynierów, handlowców i innych osób w jakikolwiek sposób związanych z sieciami lokalnymi. Adresy producentów, telefony i nazwiska pracowników z działów sprzedaży umożliwiają natychmiastowy kontakt w celu uzyskania dodatkowych informacji dotyczących np. cen i terminów dostawy.

O coraz większej popularności sieci Ethernet świadczy gwałtowny wzrost liczby producentów zamieszczonych w informatorze, wielokrotnie przewyższającej ich liczbę z pierwszego wydania.

(JZ)

[1]) Shotwell R., ed.: The Ethernet Sourcebook. 3rd Revised Edition. North-Holland, Amsterdam 1985. 554 pp. ISBN 0-444-00978-7

[2]) Metcalfe R. M., Boggs D. R.: Ethernet — distributed packet switching for local computer networks. Comm. of the ACM, Vol. 19, No. 7, 1976

[3]) Shoch J. F. et al.: Evolution of the Ethernet Local Computer Network. Computer, Vol. 15, No. 8, 1982.

Terminologia definicji formalnej Ady

W przedstawionym w poprzednim i obecnym numerze „Informatyki" artykule Dinesa Bjørnera nt. roli i zakresu formalnej definicji Ady, użyto pewnej liczby terminów zdefiniowanych niedostatecznie jasno lub nie zdefiniowanych w ogóle w treści. Jednakże, do oryginalnego wydania raportu dołączono dodatek, zawierający wyjaśnienie znaczenia kilkudziesięciu pojęć dotyczących definicji formalnej Ady. Większość z nich jest, niestety, bardzo luźno związana z treścią artykułu, część w ogóle w nim nie występuje, a pozostałe są zdefiniowane mało precyzyjnie. Dlatego w materiale przeznaczonym dla polskich czytelników zachowano tylko te terminy z dodatku, które wyraźnie występują w treści artykułu i mają dokładniej określone znaczenie. Warto przy tym pamiętać, że ich definicje w podanym brzmieniu na ogół nie są uniwersalne, lecz dotyczą użycia w tym konkretnym przypadku. Czytelnicy bliżej zainteresowani tą terminologią mogą otrzymać odbitkę oryginału dodatku (Red.).

ATESTACJA (validation) — proces objęty umową na dostawę oprogramowania, polegający na wykonaniu czynności mających na celu uzyskanie przekonania o poprawności specyfikacji, projektu lub implementacji (może polegać na przeprowadzeniu formalnego dowodu); stwierdzenie, że specyfikacja, projekt lub implementacja spełnia narzucone wymagania.

DEKOMPOZYCJA (decomposition) — transformacja specyfikacji **A** na specyfikację **B**, polegająca na tym, że funkcje (operacje) **A** są określone przez złożenie funkcji (operacji) **B**.

DOKUMENT (document) — określony i skończony zbiór zarejestrowanych informacji, wytworzony podczas opracowywania oprogramowania. Dokument może być wyrażony w języku formalnym lub nie, może być zarejestrowany środkami elektronicznymi lub innymi. Tak więc, program i specyfikacja są dokumentami w takim rozumieniu.

POPRAWNOŚĆ (correctness) — pojęcie to ma znaczenie w metodzie formalnej, która wymaga wygenerowania przynajmniej jednej pary dokumentów formalnych, **A** i **B**, tak że **A** uważa się za podstawę do wytworzenia **B**. **B** jest poprawne wtedy, gdy spełnia swój opis zawarty w **A**. Najczęściej występuje szereg par tego rodzaju **(A, B)**, **(B, C)**, **(C, D)** itd.

PROJEKT (design) — oświadczenie dostawcy oprogramowania, określające, jak będzie zaimplementowana specyfikacja. Oświadczenia tego rodzaju mogą mieć różny poziom szczegółowości.

PROTOTYP (prototype) — wykonywalny model oprogramowania, zgodny z podzbiorem (częścią) wymagań specyfikacji.

SPECYFIKACJA (specification) — wykonany przez dostawcę oprogramowania opis funkcjonalny zachowania się implementacji i procesu jej wytwarzania.

SPECYFIKACJA UKIERUNKOWANA NA MODELE (model oriented specification) — specyfikacja (lub projekt) określająca model matematyczny, tj. obiekt lub klasę obiektów, których istnienie jest ustalone.

SPECYFIKACJA UKIERUNKOWANA NA WŁAŚCIWOŚCI (property oriented specification) — specyfikacja określająca jedynie charakterystykę zewnętrzną.

TRANSFORMACJA (transformation) — proces lub wynik generowania specyfikacji formalnej **B** na podstawie specyfikacji formalnej **A**, taki, że **B** spełnia **A**, przy czym struktura wewnętrzna **B** zwykle nie jest dekompozycją struktury **A**. O ile nie zastrzeżono inaczej, transformacja obejmuje uściślenie i wzbogacenie.

TWORZENIE PROTOTYPU (prototyping) — czynność polegająca na konstruowaniu prototypu, zazwyczaj związana z przekształceniem specyfikacji lub abstrakcyjnego projektu na wykonywalny kod wysokiego poziomu.

UŚCIŚLENIE (refinement) — zachowujące strukturę wewnętrzną zdekomponowanie specyfikacji **A** na (zazwyczaj abstrakcyjny) projekt **B**, lub zdekomponowanie projektu **B** na projekt **C**.

WERYFIKACJA (verification) — dowodzenie, że transformacja, wzbogacenie lub uściślenie jest poprawne; terminem tym określa się także proces objęty umową na dostawę oprogramowania, polegający na wykonaniu czynności mających na celu uzyskanie przekonania o właściwym sformułowaniu i niezdegenerowaniu specyfikacji, projektu lub implementacji.

WZBOGACENIE (enrichment) — transformacja, która powoduje rozszerzenie funkcji; w metodzie algebraicznej lub denotacyjnej może obejmować dodanie operacji.

Oprac.: JZ

WARUNKI PRENUMERATY NA 1986 R.

Prenumeratorzy zbiorowi — jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę dokonując wpłat na blankiecie „polecenie przelewu" rozszerzonym dla potrzeb Wydawnictwa o część dotyczącą zamówienia. Blankiety te będą dostarczane przez Zakład Kolportażu.

Prenumeratorzy indywidualni — osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na blankiecie Wydawnictwa lub blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamawianych egzemplarzy oraz wartość wpłaty.
Wpłacać należy na konto NBP III O/M Warszawa 1036-7490-139-11.

Prenumerata ulgowa — przysługuje wyłącznie osobom fizycznym — członkom SNT, studentom i uczniom szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu wpłaty (przed jej dokonaniem) na wszystkich odcinkach pieczęcią Koła SNT, wyższej uczelni lub szkoły.

Sposób zamawiania prenumeraty taki sam jak dla prenumeraty indywidualnej.

Prenumerata ze zleceniem wysyłki za granicę — zamawia się tak jak prenumeratę indywidualną. Dodatkowo należy podać na blankiecie wpłaty nazwisko i dokładny adres odbiorcy. Cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa.

Przedpłaty na prenumeratę przyjmowane są w terminach:
— do 10 listopada na I kwartał, I półrocze i cały rok następny,
— do 28 lutego na II, III, IV kwartał i II półrocze,
— do 31 maja na III, IV kwartał i II półrocze,
— do 31 sierpnia na IV kwartał.

Uwaga!

Wpłaty na dwumiesięczniki przyjmowane są na okresy półroczne lub roczne.

Informacji o prenumeracie udziela — Zakład Kolportażu Wydawnictwa NOT-SIGMA, ul. Bartycka 20, 00-716 Warszawa, lub skr. poczt. 1004, 00-950 Warszawa, tel. 40-00-21 w. 249, 293, 297, 299 oraz 40-35-89 i 40-30-86.

Egzemplarze archiwalne czasopism — można nabyć za gotówkę w Klubie Prasy Technicznej w Warszawie ul. Mazowiecka 12, tel. 27-43-65 oraz w Dziale Handlowym Wydawnictwa ul. Bartycka 20 skr. poczt. 1004, 00-950 Warszawa, na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.

Cena miesięcznika INFORMATYKA została ustalona na 120 zł za numer (35 zł — cena ulgowa).

Cena prenumeraty INFORMATYKI (w złotych)					
kwartalna		półroczna		roczna	
normalna	ulgowa	normalna	ulgowa	normalna	ulgowa
360,—	105,—	720,—	210,—	1440,	420,—

Udrycki W.: MML — język komunikacji z systemem komutacyjnym

INFORMATYKA 1986, nr 6, s. 1

Charakterystyka języka przeznaczonego do komunikowania się operatora ze zautomatyzowanym systemem komutacyjnym. Omówiono genezę powstania i zastosowania języka oraz jego konstrukcję i sposób działania.

Królikowski Z.: Generowanie planów wykonywania transakcji w systemach rozproszonych baz danych (1)

INFORMATYKA 1986, nr 6, s. 4

Pierwsza część charakterystyki metody optymalizacji planów wykonywania transakcji w systemach rozproszonych baz danych. Omówiono podstawowe pojęcia i założenia metody, kryteria oceny działania systemów rozproszonych baz danych oraz podano przykład operacji łączenia relacji.

Płoski Z.: Współpraca LOGO Sinclaira ze stacją dyskietek

INFORMATYKA 1986, nr 6, s. 9

Propozycja rozszerzenia możliwości języka LOGO firmy Sinclair w warunkach korzystania z pamięci na dyskach elastycznych.

Szkaradnik Z.: Grafika żółwia w języku Forth

INFORMATYKA 1986, nr 6, s. 11

Propozycja rozszerzenia języka Forth o procedury graficzne oparte na filozofii grafiki żółwia z języka LOGO. Omówiono sposób implementacji pakietu oprogramowania graficznego.

Perycz K.: Mikrokomputerowy system operacyjny S1 (2)

INFORMATYKA 1986, nr 6, s. 14

Dokończenie charakterystyki systemu operacyjnego S1. Omówiono zmienne systemowe, woluminy i pliki, algorytmy szeregowania, komunikację i synchronizację międzyzadaniową oraz kontrolę uprawnień użytkowników.

Bjørner D.: Rola i zakres formalnej definicji Ady (2)

INFORMATYKA 1986, nr 6, s. 17

Dokończenie charakterystyki roli i zakresu formalnej definicji języka Ada. Omówiono obecnie przyjęte podejście w opracowaniu definicji oraz sformułowano wymagania, jakie powinna ona spełniać.

Zaskórski P., Kasprzak E.: Technologia wytwarzania oprogramowania użytkowego z komputerowym wspomaganiem

INFORMATYKA 1986, nr 6, s. 20

Charakterystyka nowej technologii wytwarzania oprogramowania użytkowego opartej na metodologii projektowania systemów informatycznych PSL/PSA. Podano efekty stosowania nowej technologii.

Удрыцки В.: ММЛ — язык связи с системой коммутации

INFORMATYKA 1986, № 6, с. 1

Характеристика языка, предназначенного для связи между оператором и автоматизированной системой коммутации. Описание генезиса создания и применения языка, а также его конструкции и принципа действия.

Круликовски З.: Генерирование планов выполнения сделок в системах рассеянных баз данных (1)

INFORMATYKA 1986, № 6, с. 4

Первая часть характеристики метода оптимизации планов выполнения сделок в системе рассеянных баз данных. Описание основных понятий и исходных посылок метода, критерии оценки действия систем рассеянных баз данных и пример операции объединения соотношений.

Плоский З.: Взаимодействие LOGO Sinclair со станцией гибких дисков

INFORMATYKA 1986, № 6, с. 9

Предложение относительно расширения языка LOGO фирмы Sinclair в условиях использования памяти на гибких дисках.

Шкарадник З.: Графика черепахи на языке Forth

INFORMATYKA 1986, № 6, с. 11

Предложение расширения языка Forth за счет графических процедур, основанных на философии графики черепахи из языка LOGO. Описание способа применения пакета графического программного обеспечения.

Перыч К.: Микропроцессорная операционная система С1 (2)

INFORMATYKA 1986, № 6, с. 14

Окончание характеристики операционной системы С1. Описание системных переменных, томов и файлов, алгоритмов последовательности, связи и синхронизации между заданиями, а также контроля за правами пользователей.

Бьернер Д.: Роль и область формальной дефиниции языка Ада (2)

INFORMATYKA 1986, № 6, с. 17

Окончание характеристики роли и объема формальной дефиниции языка Ада. Описание применяемого в настоящее время подхода при разработке дефиниции, а также формулировка требований, которым она должна удовлетворять.

Заскурский П., Каспшак Э.: Технология разработки прикладного программного обеспечения при помощи ЭВМ

INFORMATYKA 1986, № 6, с. 20

Характеристика новой технологии разработки прикладного программного обеспечения, основанной на методологии проектирования вычислительных систем ПСЛ/ПСА. Эффекты от применения новой технологии.

Udrycki W.: MML — a language for communication with commutation system

INFORMATYKA 1986, No. 6, p. 1

Characteristics of language for opetators communication with automatized commutation system. Genezis and application of the language, as well as its structure and operation are discussed.

Udrycki W.: MML — eine Sprache zur Kommunikation mit Kommutierungssystem

INFORMATYKA 1986, Nr. 6, S. 1

Eine Charakteristik der Sprache, die für Kommunikation des Operators mit einem Kommutierungssystem bestimmt ist. Es wurden Entstehung und Anwendungen der Sprache, sowie ihre Struktur und Arbeitsweise besprochen.

Królikowski Z.: Generation of transaction plans in distributed data base systems (1)

INFORMATYKA 1986, No. 6, p. 4

First part of characteristics of the optimalization method for transaction plans generation in distributed data base systems. Basic notions and assumptions of the method, criteria for evaluation of distributed data base systems operation are discussed and an example of relation merging is presented.

Królikowski Z.: Generierung der Transaktionausführungspläne in verteilten Datenbanksystemen (1)

INFORMATYKA 1986, Nr. 6, S. 4

Erster Teil einer Charakteristik von Optimierungsmetrode der Transaktionsplänеherstellung in verteilten Datenbanksystemen. Es wurden Grundbegriffe und Vorausetzungen der Methode, Beurteilungskriterien der Wirkung von verteilten Datenbanksystemen besprochen, sowie ein Beispiel der Relationverbindungsoperation angegeben.

Płoski Z.: Sinclair's LOGO cooperation with floppy disc station

INFORMATYKA 1986, No. 6, p. 9

A proposal for Sinclair's LOGO possibilities extending under circumstances of floppy disc attachment.

Płoski Z.: Zusammenarbeit von Sinclair's LOGO mit einer Diskettenstation

INFORMATYKA 1986, Nr. 6, S. 9

Ein Vorschlag von Erweiterung der LOGO-Sprache unter Bedingungen der Ausnutzung von Diskettenspeicher.

Szkaradnik Z.: Turtle graphics in Forth

INFORMATYKA 1986, No. 6, p. 11

A proposal for Forth language extending with graphic procedures based on turtle graphic philosophy from the LOGO-language. The way of graphic package implementation is discussed.

Szkaradnik Z.: Schildkrötegraphik in der Forth-Sprache

INFORMATYKA 1986, Nr. 6, S. 11

Ein Vorschlag der Erweiterung von Forth-Sprache mit graphischen Prozeduren, die auf der Schidkrötegraphikphilosofie beruhen. Es wurde Implementierungsweise des graphischen Softwarepakets besprochen.

Perycz K.: S1 — a microcomputer operating system (2)

INFORMATYKA 1986, No. 6, p. 14

Conclusion of the S1 operatng system characteristics. System's variables, volumes and files, arranging algorithms, communication and synchronization between tasks, as well as user's authorization checking are discussed.

Perycz K.: S-1 — Microcomputerbetriebssystem (2)

INFORMATYKA 1986, Nr. 6, S. 14

Beendigung einer Charakteristik von dem S1-Betriebssystem. Es wurden Systemsvariablen, Volumen und Dateien, Ordnenalgorithmen, Aufgabenkommunikation und -synchronisation, sowie Kontrolle der Benutzerberechtigungen besprochen.

Bjørner D.: Role and scope of Ada formal definition (2)

INFORMATYKA 1986, No. 6, p. 17
Conclusion of characteristics of the Ada formal definition. Actual approach in the definition elaboration is discussed and requirements, witch should it fulfil, are formulated.

Bjørner D.: Role und Bereich der formellen Definition von Ada (2)

INFORMATYKA 1986, Nr. 6, S. 17

Beendigung einer Charakteristik von Rolle und Bereich der formellen Definition von Ada-Sprache. Es wurde die heutige Einstellung zur Definitionsbearbeitung besprochen und Forderungen, welche die Definition erfüllen sollen, formuliert.

Zaskórski P., Kasprzak E.: Computer assisted technology for production of application software

INFORMATYKA 1986, No. 6, p. 20

Characteristics of the new technology for production of application software, based on the PSL/PSA design methodology. Results of the application of this new technology are presented.

Zaskórski P., Kasprzak E.: Technologie der Anwendungssoftwareherstellung mit Computerunterstützung

INFORMATYKA 1986, Nr. 6, S. 20

Eine Charakteristik von neuen Technologie der Anwendungssoftwareherstellung, die auf der PSL/PSA-Projektierungsmethodologie für EDV-Systeme basiert. Es wurden Anwendungseffekte der neuen Technologie angegeben.

COMPUTER STUDIO KAJKOWSCY

PROFESJONALNE OPROGRAMOWANIE MIKROKOMPUTERÓW

ul. Balladyny 3B, 81-524 Gdynia, tel.: 29-00-18, 24-01-50

Już od czterech lat działamy na rynku oprogramowaniu mikrokoputerów w Polsce, tworząc systemy znajdujące szerokie zastosowanie w zarządzaniu, finansach, księgowości i gospodarce materiałowej — narzędzia wspomagające pracę programistów, dyrektorów, księgowych i sekretarek.

Nasze programy sprawdziły się już w ponad 500 przedsiębiorstwach i instytucjach.

Dzięki wysiłkowi najlepszych fachowców jesteśmy w stanie co miesiąc przedstawić Wam nowy produkt o najwyższym standarcie światowym. Zaletą naszego oprogramowania jest możliwość wzajemnej współpracy poszczególnych systemów i tworzenia z nich pakietów o specjalnej, dostosowanej do potrzeb klienta, strukturze.

Jednocześnie firma nasza nieodpłatnie zapewnia serwis gwarancyjny i wymianę zakupionych już programów na najnowsze, udoskonalone ich wersje. Wszystkie nasze systemy są łatwe w użyciu, opracowane w języku polskim i zaopatrzone — dla wygody klienta — w wyczerpującą dokumentację.

Dla zainteresowanych oferujemy również pomoc we wdrożeniu naszych pakietów i szkolenie w zakresie ich obsługi.

Współczesny świat wymaga szybkiej i dokładnej informacji, a my chcemy pomóc w jej uzyskaniu — dlatego jesteśmy pewni, że w Twoim przedsiębiorstwie znajdą zastosowanie oferowane przez nas systemy.

	Nazwa programu	8-B	16-B
	Oprogramowanie użytkowe		
1	**BANK-DANYCH CSK** — System zarządzania bazą danych	+	+
2	**TABPLAN CSK** — Komputerowy arkusz kalkulacyjny	+	+
3	**TEKST CSK** — Pakiet redagowania tekstów	+	+
4	**TRANSCOM CSK** — Program komunikacji z ODRĄ	+	+
5	**TRANSCOM/M CSK** — Program komunikacji między mikrokomputerami	+	+
6	**BANK-GSK** — Graficzny system komunikacji z bazą danych	—	+
7	**BGRAF CSK** — Pakiet grafiki prezentacyjnej	—	+
8	**FK CSK** — System finansowo-księgowy	+	+
9	**EM CSK** — System ewidencji materiałowej	+	+
10	**PL_TEKST** — Pakiet redagowania tekstów (polskie znaki)	—	+
11	**CSK SOFT#1** — Oprogramowanie pomocnicze — BANK	—	+
12	**BANK-TP CSK** — Komunikacja między BANK CSK i TABPLAN	+	+
13	**OZ CSK** — Odtwarzanie zbiorów danych BANK CSK	+	+
	Oprogramowanie systemowe		
14	**SOMIK** — Rozbudowa systemu operacyjnego CP/M 2.0		
15	**W/SYS16 CSK** — System operacyjny wielozadaniowy — wielokonsolowy		
16	**GKS CSK** — Pakiet procedur graficznych wg normy GKS		

+ tak; — nie

Oprogramowanie CSK działa na wszystkich mikrokomputerach 8-bitowych z systemem operacyjnym CP/M oraz na komputerach 16-bitowych kompatybilnych z IBM PC/XT/AT, dostępnych w Polsce.

NOWOŚĆ!

WIELOKONSOLOWY, WIELODOSTĘPNY MIKROKOMPUTEROWY SYSTEM OPERACYJNY

EO/81/K/86

amepol

PRZEDSIĘBIORSTWO ZAGRANICZNE

Przedsiębiorstwo Zagraniczne AMEPOL

oferuje

PÓŁPRZEWODNIKOWE PAMIĘCI OPERACYJNE

do komputerów:

- **ODRA 1305**

pojemność od 64 k do 2 M

- **R – 32**

pojemność od 256 k do 16 MB

do komputerów:

- **MERA 400**

pojemność od 64 k do 1 M

- **MERA 9150 (SEECHECK)**
- **PDP 11 – SM 4**

Pamięci wykonane są na elementach VLSI produkcji zachodniej.
Na powyższe pamięci udzielamy dwuletniej gwarancji.
Terminy dostawy według życzeń klienta.

PZ AMEPOL oferuje ponadto:

- do komputera **ODRA 1305** skaner oraz multiplekser
- do minikomputera **MERA 400** procesory peryferyjne **PLIX** i **MULTIX** oraz moduł inicjatora

Szczegółowych informacji udziela:

Przedsiębiorstwo Zagraniczne
AMEPOL

Zakład Elektroniki i Aparatury Medycznej
Plac Żelaznej Bramy 1
00-136 Warszawa
tel.: 20-34-75, 20-45-05
teleks: 812539 apol pl

EO/461/K/86